

Analysing and visualising data sets of cybercrime investigations using Structured Occurrence Nets



Talal Alharbi
School of Computing
Newcastle University

A thesis submitted for the degree of
Doctor of Philosophy

November 4, 2020

Dedication

This work is dedicated to my family, my supervisor and my friends..

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, foot notes, tables and equations and has fewer than 150 figures.

Talal Alharbi
November 4, 2020

Acknowledgements

First of all, praises and thanks to Allah, who has granted me countless blessings, the patience and perseverance during this research project, and indeed, throughout all my life.

I gratefully acknowledge the deepest gratitude to my supervisor, *prof. Maciej Koutny*, for his great guidance and wise supervision. Without his grate continuing support and encouragement, this research would never be a reality.

Also, I would like to thank Prof. Brian Randell for his help, guidance and support. Moreover, I cannot forget to thank Dr Anirban Bhattacharyya and Dr Bowen Li for support and help me during my journey.

In addition, I would like to thank my family especially Mom and Dad, for the continuous support they have given me throughout my time, friends and indeed my love *Seham* who gave me all the support I needed during this journey.

Also,I would like to thank Dr. Majed Alhaisoni for his support, help and encourage me.

Abstract

Structured Occurrence Nets (SONs) are a Petri net based formalism for portraying the behaviour of complex evolving systems. As a concept, SONs are derived from Occurrence Nets (ONs). SONs provide a powerful framework for evolving system analysis and are supported by the existing SONCraft toolset. On the other hand, modelling of cybercrime investigations has become of interest in recent years, and large-scale criminal investigations have been considered as complex evolving systems. Right now, they present a significant challenge for police investigators and analysts. The current thesis contributes to addressing this challenge in two different ways: (i) by presenting an algorithm and an implemented tool that visualise data sets using maximal concurrency; and (ii) by detecting DNS tunnelling through a novel SON-based technique and tool. Moreover, the theoretical contribution of this thesis focuses on model extensions and abstraction; in particular, it introduces a new class of SONs based on multi-coloured tokens.

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Research Question	3
1.3	Contributions	5
1.4	Thesis Structure	6
1.5	List of Publications	7
2	Structured Occurrence Nets, Visualisation and DNS Background	8
2.1	Summary	8
2.2	Introduction	9
2.3	Structured Occurrence Nets (SONs)	10
2.3.1	Occurrence Nets	10
2.3.2	Communication Structured Occurrence Nets	13
2.3.3	Behavioural Structured Occurrence Nets (BSONs)	17
2.3.3.1	Synchronous Cycles and Asynchronous Cycles	18
2.3.4	Temporal Structured Occurrence Nets	19
2.3.5	Time in SONs	20
2.4	SONS and Alternative Modelling Techniques	21
2.4.1	Attack Graphs	21
2.4.2	SON vs ML vs Attack Graph	22
2.5	Visualisation	23
2.5.1	Importance of visualisation	24
2.5.2	Visualisation stages	24
2.5.3	Visualisation techniques	25
2.5.3.1	Technical Approach	26
2.5.3.2	Interaction techniques	28
2.6	Domain Name System (DNS)	30
2.6.1	Name servers	32

2.6.2	Location transparency	32
2.6.3	DNS records	33
2.6.4	DNS Tunnelling	33
2.6.5	Existing Detect DNS Tunnelling	35
2.6.5.1	Traffic Analysis	35
2.6.5.2	Payload Analysis	38
2.7	Conclusion	40
3	Visualising Data Sets in Structured Occurrence Nets	41
3.1	Summary	41
3.2	Introduction	42
3.3	Policy Driven Visualisation	42
3.4	Loading data sets in SONCraft	43
3.5	Proposed solution	44
3.6	Policy-driven visualisation algorithm	45
3.7	Implementation and test results	50
3.8	Limitations of the proposed Visualisation Approach	55
3.9	Concluding remarks	55
4	Domain Name System (DNS) tunnelling detection using Structured Occurrence Nets (SONs)	56
4.1	Summary	56
4.2	Introduction	57
4.3	Authoritative name server	58
4.4	DNS tunnelling tools	60
4.5	Preprocessing datasets	61
4.5.1	Mixing data	63
4.6	Proposed solution	64
4.6.1	Detecting DNS tunnelling using SONs	64
4.6.2	Detection DNS tunnelling algorithm	65
4.6.2.1	The calculated threshold	65
4.7	Result testing and evaluation	68
4.8	Concluding remarks	70

5	Visualisation and Abstract Conditions (Extensions)	71
5.1	Summary	71
5.2	Introduction	71
5.3	Big Data	73
5.4	Petri Nets (PNs)	74
5.5	Coloured Petri Nets (CPNs)	75
5.6	Extensions	76
5.7	Examples of coloured tokens in SONs	82
5.8	Concluding remarks	89
6	SONCraft: A Plug-in for Loading Large Amount of Data	90
6.1	Summary	90
6.2	Introduction	90
6.3	Loader algorithm	91
6.3.1	The algorithm	92
6.4	Plugin design and implementation Design	94
6.4.1	Graphical User Interface design	94
6.4.2	Data retrieval Back-end design	95
6.5	Evaluating the SONCraft Plug-in	99
6.6	Conclusion and Future Work	100
7	Conclusion and Future work	101
7.1	Conclusion	101
7.2	Future work	102
7.2.1	Visualisation of behavioural abstraction through maximal concurrency policy	102
7.2.2	Detection of multi-attacks and use of different DNS tunnelling tools	103
7.2.3	Investigation of large data sets using coloured tokens of SONs	104
7.2.4	Data mining and SONs	104
	Bibliography	105

List of Figures

2.1	An occurrence net.	12
2.2	A phase decomposition of an occurrence net.	13
2.3	Asynchronous and synchronous communication.	14
2.4	Behavioural Structured Occurrence Nets.	17
2.5	Using SONs features in a crime scenario.	18
2.6	Temporal Structured Occurrence Net.	20
2.7	Time in SON.	21
2.8	Visualisation process [69].	24
2.9	How DNS Resolves Names and IP Addresses (the labels on arcs indicate successive steps of DNS protocol).	32
2.10	DNS tunnelling.	34
3.1	Overlapping and out-of-order ONs.	43
3.2	Removing overlapping of ONs.	44
3.3	Maximally concurrent representation of events.	45
3.4	Special kind of cycles in SONs.	47
3.5	The case of asynchronous communication.	48
3.6	The standard Result.	48
3.7	The result after detecting asynchronous communication.	50
3.8	Retrieve Big Data menu.	50
3.9	First test result.	51
3.10	Second test result.	52
3.11	Third test result.	53
3.12	Fourth test result.	54
4.1	DNS tunnelling steps.	59
4.2	Normal DNS packet SON model	61
4.3	Normal and Abnormal ONs	64
5.1	Normal and Abnormal ONs.	72

5.2	Petri Net example.	75
5.3	Coloured tokens in simple example (Step 1).	82
5.4	Coloured tokens in simple example (Step 2).	82
5.5	Coloured tokens in simple example (Step 3).	83
5.6	Coloured token in ON (Step 1).	83
5.7	Coloured token in one ON (Step2).	83
5.8	Coloured token in one ON (Step 3).	84
5.9	Coloured token in one ON (Step 4).	84
5.10	Coloured token in one ON (Step 5).	84
5.11	Coloured token in one ON (Step 6).	84
5.12	Coloured token in one ON (Step 7).	85
5.13	Coloured token in one ON (Step 8).	85
5.14	Coloured tokens in CCSON (Step 1).	85
5.15	Coloured tokens in CCSON (Step 2).	86
5.16	Coloured tokens in CCSON (Step 3).	86
5.17	Coloured tokens in CCSON (Step 4).	87
5.18	Four ONs represent normal packets of DNS and one ON represents Local server.	87
5.19	Abstracting large model to small one.	88
5.20	Abstracting large model to small one.	89
6.1	Conversion of CSV file data into ONs.	92
6.2	Graphical user interface classes.	94
6.3	Import SON Data tool.	95
6.4	Maximal ONs events tool.	95
6.5	Data retrieval feature class diagram.	96
6.6	The code.	97
6.7	Part two of the code	98
7.1	SON Model for Normal DNS protocol	103

List of Tables

4.1	Terminology used in the discussion of DNS tunnelling.	58
4.2	DNS Packets	62
4.4	DNS Tunnelling Packet	62
4.3	Normal DNS Packet	62
4.5	Mixed Packets (normal and abnormal packets)	63
4.6	Table of results	70
6.1	CSV file data	92

Chapter 1

Introduction

1.1 Introduction

Today's increasingly complex computing, commercial and societal systems are becoming more of a challenge for researchers, investigators, and analysts. Through the ongoing processes of discovery and analysis, the latter ultimately end up contributing to the evolution of these systems. As far as a 'complex system' goes, an attempt to provide an exhaustive definition may be daunting. However, a broad definition has been reached, according to which a complex system generally implies an intricate structure, and generally consists of a number of subsystems that interact with one another and are subject to various external stimuli. Moreover, a complex system is usually characterised by constant evolution, meaning it is not created within a single moment, but rather it is the result of evolution occurring over time. As an example, a complex evolving system may be biological, or informational in nature. One specific example of such a system is a cybercrime modelling and investigation system. Each such system consists of several subsystems that interact with one another. This process of communication between them may be synchronous or asynchronous.

Cybercrime modelling and investigation systems use various methods and tools supporting investigations, ranging from complex networks to partial differential equations, and statistical analyses. The modelling and investigation of large-scale, criminal cases are now increasingly considered as an instance of complex evolving systems, and

their complexity makes them a significant challenge for police investigators. The visualisation of big data sets in a user-friendly and suitable way makes it possible for investigators to have a clearer image of the events which occurred, allowing them to link crime events and places with the purpose of getting a clearer representation that could simplify crime detection. In addition, the result of improved visualisation techniques could very well constitute an essential step in preventing events such as that from occurring in the first place. Cybercrime events are part of the criminal investigation process and constitute one of the core organisational activities for police services [24]. The current existing literature [6] and suggested practices indicate large scale investigations can be conducted thanks to the aforementioned mechanisms. Unfortunately, there are few tools that can provide support in this regard. Considering the current academic context, along with the aforementioned considerations regarding the evolution of complex cybercrime analysis, a number of theoretical focus areas are relevant [42]. One such area [42, 59, 57] includes Structured Occurrence Nets (SONs). The research conducted in this thesis will concern itself with the theoretical and practical aspects of SONs. SONs [57] are a Petri net based formalism for portraying the behaviour of complex evolving systems. The concept of SONs is derived from that of Occurrence Nets (ON). An occurrence net is a directed, acyclic graph that represents causal and concurrent information regarding a single execution of a system [58]. Within a SON, multiple ONs are linked with each other through various types of formal relationships. They are intended to record information about: (i) the actual or projected behaviour of complex systems, as they interact and evolve; and (ii) the evidence that is being gathered and analysed, regarding the past behaviour of complex evolving systems. Furthermore, SONs have the advantage of supporting various representations of the behaviour of a complex evolving system [59, 57]. A particularly useful way of using SONs is within sophisticated investigations. The significance of SONs comes from their structuring and different forms of abstraction,

which may considerably reduce the complexity of modelling and analysis, and the fact that they provide a direct means of modelling evolving systems.

1.2 Research Question

The aim of the research programme presented in this thesis was to extend the SON framework, so as to improve both the handling and visualisation of data sets involved in cybercrime investigations. As such, the main research question was: “Can tools which use SONs be used to analyse big data cybercrimes?”, followed by an immediate additional question: “Can SONs cope with a large volume of records, stored in specific data sources?” In more detail, the question was whether or not SONs and SON-based tools, such as SONCraft, can provide a new approach which can be used to analyse large cybercrime data sets, by coping with a considerable volume of data which can be stored either online or offline. To address this question, a methodology was developed and implemented, involving both theoretical and practical research. The main goals of this methodology were as follows:

Goal I: Enhancing visualisation of SONs

To design and implement an algorithm to assist in visualising large data sets in a helpful, user-friendly and understandable way. The proposed approach was the use of semantically driven visualisation, which adopts maximal concurrency (a step execution policy in the sense of [9]) to provide structured displays. As a result, the new approach should lead to an effective display of complex SONs.

Working on Goal I

SONCraft was evaluated from the perspective of interaction techniques, as issues were to be expected when inputting data into SONCraft (in particular, after allowing the tool to automatically accept data sets). Some of these issues, such as overlapping ONs, could be addressed by using maximal concurrency.

Concerning the implementation of new visualisation technique, it focused on two main purposes of data analysis, namely understanding and exploration. The thesis addressed the challenges resulting from the overlapping and out-of-order placement of ONs, and proposes a novel solution to deal with this issue, based on a step-execution policy. A SON plugin implementing the resulting algorithm was developed.

Goal II: Detecting cybercrime using SONs

To apply SONs and their features in order to detect real-data cybercrime cases (DNS Tunnelling) in the event of an actual attack.

Working on Goal II

The research went through various stages, starting with data collection (capturing real data from a local DNS server) and an extensive study of literature pertaining to the subject. To simulate a real scenario of DNS server behaviour, the collected data consisted of two main parts, the first of which included data collected during the attack experiment, and real DNS traffic data. To help with the detection of an attack, a solution based on a new algorithm was proposed, along with a tool implementation.

Goal III: Enhancing SON theory

To extend the theoretical foundations of SONs, making it possible to abstract large ONs via multiple-tokens.

Working on Goal III

As a result of the experiments conducted for Goal II, large SON models were generated. Since such models were difficult to display on computers screens, a new class of SONs was proposed. This definition extension will allow users to use multiple tokens in a condition (local state) instead of a single token (as in the current SON framework). This extension will also allow users to visualise

large-model SONs more effectively.

Goal IV: Evaluating new algorithms and tools

To thoroughly evaluate the developed algorithms and tool implementation, demonstrating their validity and sustainability for big datasets.

Working on Goal IV

Tools developed when working on Goals I and II were applied to various cyber-crime cases, with the purpose of assessing and evaluating the manner in which SONs can be helpful for ‘real-life’ scenarios.

1.3 Contributions

The current thesis made the following key contributions to the improvement of SON visualisation and cybercrime investigations analysis:

1. Visualisation algorithms were designed and developed as a Java plugin within the SONCraft toolkit (based on the WorkCraft platform). These algorithms were developed and implemented using the concept of maximal concurrency in order to avoid overlapping and the out-of-order placement of ONs.
2. By using SONs in a “real-life”, cybercrime scenario, a plugin that detects DNS tunnelling has been developed and implemented.
 - (a) Preprocessing Datasets (DNS packets): real data from servers was collected, with the purpose of allowing SONs to accept this data. A script was created, with the aim of dealing with DNS packets and automatically reprocess them.
 - (b) A script which helps in mixing two types of data was designed and implemented. These two data types include the normal data (normal packet

DNS data) and the attack data. The purpose was to ensure the script works in a real-life scenario.

(c) An algorithm which detects DNS attacks through the use of SONs was designed and implemented.

3. For abstracting a large model of SONs, an extension of SONs was proposed. It allows multiple tokens to be present in the conditions of the underlying network structure.

4. Design and implement open-source SONCraft plug-in, which is used help users in automatically load large amounts of data.

1.4 Thesis Structure

Chapter 1. Describes the background, knowledge and motivation behind the work carried out in this thesis. Moreover, the chapter describes the contribution of the research programme reported in this thesis towards the general advancement of the relevant field and the publication of its results.

Chapter 2. Background and related work - This chapter focuses on the conducted background research and the review of the existing literature and the current “state-of-art” when it comes to Structured Occurrence Nets (SONs) and their features (Behavioural SONs, Communication SONs, Temporal Abstraction SONs, and Time SONs). The chapter also discusses DNS and “DNS Tunnelling” as concepts and describes the main challenges associated with DNS tunnelling. Also, several techniques for DNS tunnelling detection are reviewed.

Chapter 3. Visualisation - This chapter describes visualisation as a background concept, its framework, and the importance of visualising large data sets in

SONs. Moreover, the chapter places focus on the algorithms which provide a solution to the handling of large data visualisation issues within SONs.

Chapter 4. DNS Tunnelling - This chapter focuses on illustrating the challenges within DNS tunnelling and how these challenges can be surpassed by using SONs to detect potential attacks.

Chapter 5 This chapter presents and describes an extension of SONs that allows multiple tokens within the same conditions.

Chapter 6 This chapter provides a plug-in to Load Big Data in Structured Occurrence Nets. .

Chapter 7. Concludes the description of research and reflects on its results. The limitations of the current research and its results are illustrated, along with several directions for future work and research.

1.5 List of Publications

1. Alharbi, T. and Koutny, M., 2019. Domain Name System (DNS) Tunnelling Detection using Structured Occurrence Nets (SONs). In Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2019). Newcastle University.
2. Alharbi, T. and Koutny, M., 2018. Visualising Data Sets in Structured Occurrence Nets. In PNSE@ Petri Nets/ACSD (pp. 121-132).
3. Li, B., Randell, B., Bhattacharyya, A., Alharbi, T. and Koutny, M., 2018, June. SONCraft: A Tool for Construction, Simulation, and Analysis of Structured Occurrence Nets. In 2018 18th International Conference on Application of Concurrency to System Design (ACSD) (pp. 70-74). IEEE.

Chapter 2

Structured Occurrence Nets, Visualisation and DNS Background

2.1 Summary

SONs are Petri net based formalism for portraying the behaviour of complex evolving systems. Their concept is originally grounded on an occurrence net (ON). SONs provide strong results in terms of the analysis of an evolving system, as the analysis of the current SONCraft framework reveals.

This chapter will focus on background material as well as the motivation of the work carried out in this thesis. Section 2.3 focuses on an overview of structured occurrence nets (SONs) and its features. Section 2.4 discusses the importance of visualisation, its background, and visualisation techniques. Then we give an overview of the domain name system (DNS) and discuss DNS tunnelling in Section 2.5. Also, we review the existing methods for detecting DNS tunnelling. The last section contains concluding remarks.

2.2 Introduction

SONs [42, 57] are a Petri net based formalism for portraying the behaviour of complex evolving systems. Their concept is originally grounded on an occurrence net (ON), which is a directed, acyclic graph illustrating the causality and concurrency information concerning a single execution of a given system [42, 57]. Within a SON, multiple ONs are associated with each other through various types of formal relationships. They are intended to record information about: (i) the actual or imagined behaviour of complex systems, as they interact and evolve; and (ii) evidence being gathered and analysed regarding the past behaviour of complex evolving systems. SONs have the advantage of supporting various representations of the behaviour of a complex evolving system [42]. A particularly useful way to use SONs would be as part of a sophisticated crime or accident investigation support, and at the moment there is a general lack of investigation support systems based on SONs. The significance of SONs results from their structuring, as it reduces the complexity when compared to any other equivalent representation. They also provide a direct means of modelling behaviour of evolving systems.

This chapter will present SON background information and main features. Firstly, occurrence nets will be introduced. Afterwards, we will illustrate communication within SONs, covering both synchronous and asynchronous communication. Then, the behavioural and temporal dimensions within a SON are discussed. Subsequently, verification and time in SONs are presented. Finally, the chapter is concluded.

2.3 Structured Occurrence Nets (SONs)

2.3.1 Occurrence Nets

Occurrence nets (ONs) are acyclic Petri nets which can be used to record dependencies between states and events in executions of concurrent systems. The main relations between events in ONs are *causality* and *concurrency*. Each occurrence net ON has three main components: *conditions* C , *events* E , and a binary *flow relations* F . Conditions and events are the nodes in the graphical representation of ON , and each flow relation tuple is represented by an *arc*. In brief, conditions within ON are linked to events via arcs (the sources are conditions and the destinations are events). Also, arcs link events to conditions. For a given node x , the set of input nodes is represented by $\bullet x$, and the set of output nodes is represented by x^\bullet . The initial state of ON contains the conditions with empty inputs, and the final state consists of conditions with empty outputs. Regarding ON conditions, each one has at most one input event, and at most one output event [45]. Moreover, for each event within ON , there is at least one input condition and at least one output condition. A global state (or marking) within ON is a maximal set comprising pairwise concurrent (i.e., there is no directed path between them) conditions [42]. In general, ONs are cyclic repetitions for the same events, which is recorded as a new element.

Definition 2.3.1 (ON) *An occurrence net is a triple $ON = (C, E, F)$, where C and E are finite disjoint sets of respectively conditions and events (collectively referred to as the nodes), and $F \subseteq (C \times E) \cup (E \times C)$ is the flow relation. The inputs and outputs of a node x are respectively defined as $\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$.¹ For a set of nodes $X \in (C \cup E)$, we respectively denote by $\bullet X$ and X^\bullet the sets of all inputs and outputs of the nodes in X . It is also assumed that*

¹In this thesis, sometimes we will, for the purpose of clarity, use the notations $pre(x)$ and $post(x)$ instead of a ‘dot’ to represent inputs and outputs.

the following are satisfied:

- i. for every condition c there is at most one event e such that $(e, c) \in F$, and at most one event f such that $(c, f) \in F$;
- ii. for every event e there is at least one condition c such that $(c, e) \in F$, and at least one condition d such that $(e, d) \in F$; and;
- iii. the relation $Pre_{ON} = (F \circ F) \upharpoonright_{(C \times C)}$ is acyclic (in other words, its transitive closure is irreflexive), and so ON forms an acyclic graph and F^+ is a partial order relation.

A marking (or state) of ON is a set of conditions M . A cut of ON is a maximal set M such that $(M \times M) \cap F^+ = \emptyset$. The initial marking/cut is $M_0^{ON} = \{c \in C \mid \bullet c = \emptyset\}$, and the final marking/cut is $M_{Fin}^{ON} = \{c \in C \mid c \bullet = \emptyset\}$.

A step of ON is a nonempty set of events U such that $(\bullet e \cup e \bullet) \cap (\bullet f \cup f \bullet) = \emptyset$, for all distinct $e, f \in U$.

The execution of ON proceeds by firing sets of events (steps). The next definition shows conditions under which a marking enables a step and how firing this step changes the current marking.

Definition 2.3.2 (ON firing rule) Let $ON = (C, E, F)$ be an occurrence net, M be a marking, and U be a step of ON .

1. U is ON -enabled at M if $\bullet U \subseteq M$.

2. If U is ON -enabled at M , then U can fire and produce a new marking M' given by $M' = (M \setminus \bullet U) \cup U \bullet$.

This is denoted by $M[U]_{ON} M'$.

3. A step sequence is a sequence of steps $U_1 \dots U_k$ such that there are markings

M_0, M_1, \dots, M_k such that $M_0 = M_0^{ON}$ and $M_{i-1}[U_i]_{ON} M_i$ for $i = 1, \dots, k$.²

²One can show that all the M_i 's are cuts of ON .

As shown in Figure 2.1, conditions are represented by circles and events by boxes. The initial marking in this model is $\{c0\}$ which has a token within the starting condition. Five cuts are present in this model, namely: $\{c1, c2\}$, $\{c1, c4\}$, $\{c3, c4\}$, $\{c2, c3\}$, and $\{c5\}$. A possible step sequence is $y = \{e0\}\{e1, e2\}\{e3\}$. This marking sequence starts at $M0 = \{c0\}$ and finishes with $\{c5\}$.

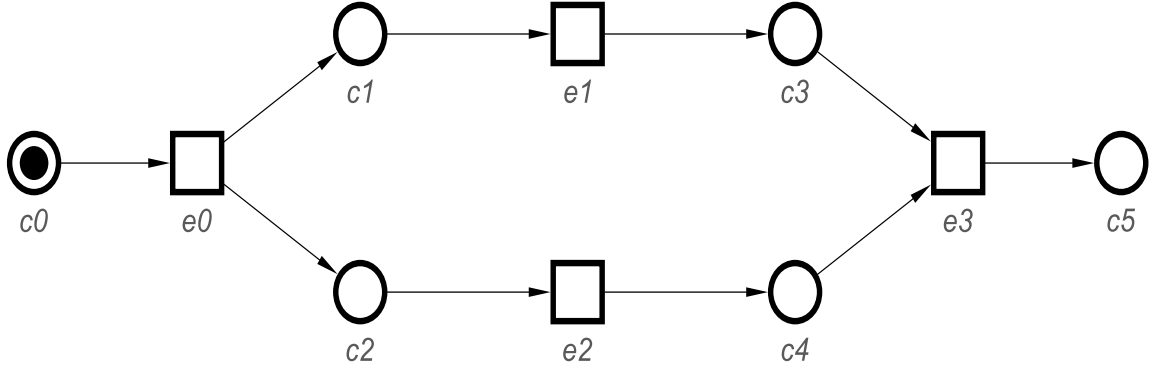


Figure 2.1: An occurrence net.

A *phase* of ON is a non-empty set of conditions $\pi \subseteq C$ such that the set $Min_\pi \subseteq \pi$ of the minimal conditions of π (w.r.t. F^+) is a cut; the set $Max_\pi \subseteq \pi$ of the maximal conditions of π (w.r.t. F^+) is a cut; and π comprises all conditions $c \in C$ for which there are $b \in Min_\pi$ and $d \in Max_\pi$ satisfying $(b, c) \in F^*$ and $(c, d) \in F^*$. Moreover, a *phase decomposition* of ON is a sequence $\pi_1 \dots \pi_m$ of phases of ON such that $M_0^{ON} = Min_{\pi_1}$, $Max_{\pi_i} = Min_{\pi_{i+1}}$ (for $i \leq m - 1$), and $Max_{\pi_m} = M_{Fin}^{ON}$. A *block* of ON is a non-empty set $B \in (C \cup E)$ such that $B \cap C = \bullet(B \cap E) \cap (B \cap E)^\bullet$ and $(\bullet B \setminus B) \times (B^\bullet \setminus B) \subseteq F^*$ [60, 45].

Each phase is a fragment of ON beginning with a cut and ending with a cut which follows it in the causal sense, including all the conditions occurring between these cuts. A *phase decomposition* is a sequence of phases from the initial cut to the final cut, and whenever one phase ends, its maximal cut is the starting point of the successive one (minimal cut) [45, 60]. A block represents a contiguous fragment of activity within the behaviour represented by ON . Intuitively, a block is a set of nodes

in which all conditions are internal to the block, that is, the input and output events of each condition must belong to the block; and all block ending events are causally linked to all starting events. Note that a single event can constitute a block [45, 60].

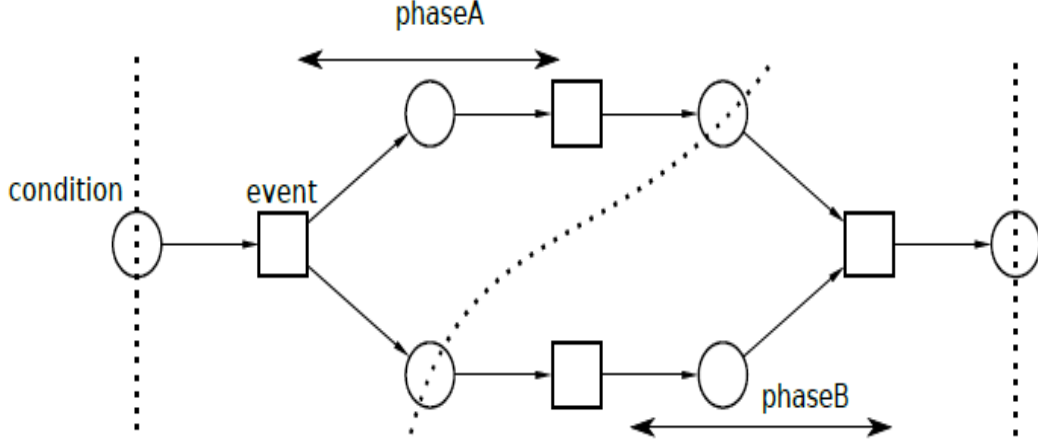


Figure 2.2: A phase decomposition of an occurrence net.

Occurrence nets describe causality and concurrency relations between events. There is also the possibility to derive occurrence nets from historical data such as log files, so as to directly illustrate a system's execution history [60]. ON modelling can represent computing systems and their histories, but they can also include components and systems involving people and even natural processes, such as crime investigations or cybercrime scenarios.

2.3.2 Communication Structured Occurrence Nets

There are two types of abstract relations which show communications between different ON subsystems. Communication within SONs is explained [42, 45] through separate occurrence nets (ONs); these proceed concurrently and occasionally communicate with each other. For instance, systems can be represented as sets of ON components, i.e., each ON is a subsystem, interacting with each other. The events

in each separate ON are responsible for associations between ONs; these associations usually occur via a special element called channel place. Communication in SONS is of two types: asynchronous and synchronous [42, 45].

Figure 2.3 shows a communication structured occurrence net CSON which consists of two occurrence nets, namely $ON1$ and $ON2$. Asynchronous communication which is represented through the dashed arc between two events in different ONs, e.g., $e0$ and $e2$.

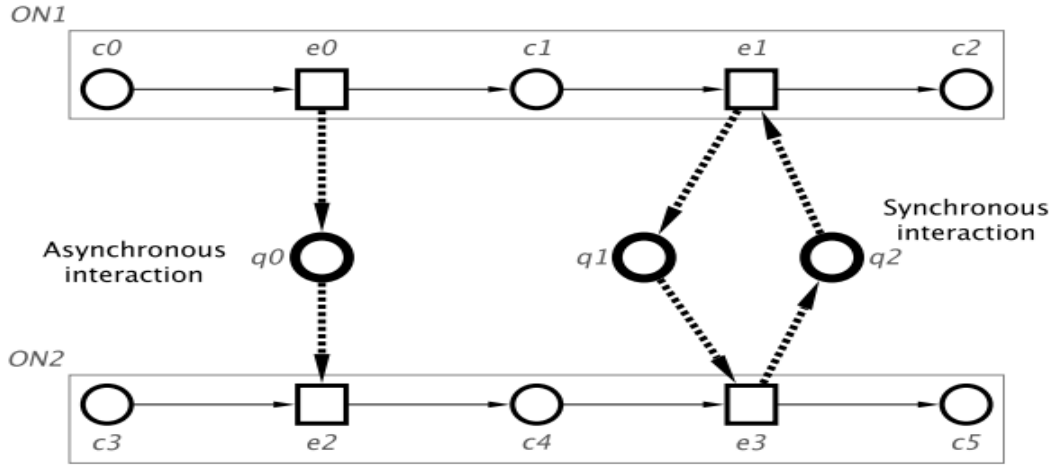


Figure 2.3: Asynchronous and synchronous communication.

The second type of communication within SONS is synchronous communication which is represented by the arcs between events $e1$ and $e3$ via channel places $q1$ and $q2$ [42, 45]. Thus, in any execution consistent with the causal relationships captured by (CSON), $e0$ will never be executed after $e2$, although the two events can be executed simultaneously, whereas $e1$ and $e3$ will always be executed simultaneously. SONS are underpinned by causal structures which extend causal partial orders with additional ordering, called weak causality. Two events ordered in this way can be executed in the order given or simultaneously. Moreover, if two events are weakly ordered in both directions (this means, in particular, that weak causality is not assumed to be acyclic), then they can only be executed simultaneously. In SONS, weak causality

results from passing tokens through channel places, whereas the non-channel places introduce the standard causality, as in ONs. In Figure 2.3, e_0 weakly causes e_2 , and the events e_1 and e_3 form a weak causality cycle.

Definition 2.3.3 (CSON) A communication structured occurrence net [45, 60] (CSON) is a tuple

$$\text{CSON} = (ON_1, \dots, ON_k, Q, W)$$

such that $ON_i = (C_i, E_i, F_i)$ for $i = 1, \dots, k$ are occurrence nets (below we denote by $\mathbf{C} = \bigcup_{i=1}^k C_i$, $\mathbf{E} = \bigcup_{i=1}^k E_i$ and $\mathbf{F} = \bigcup_{i=1}^k F_i$ their conditions, events and arcs); Q is a set of channel places; and $W \subseteq (\mathbf{E} \times Q) \cup (Q \times \mathbf{E})$ are the arcs between the channel places and events. It is further assumed that:

1. The ON_i 's and Q are mutually disjoint.
2. The sets of input and output events of $q \in Q$,

$$\bullet q = \{e \in \mathbf{E} \mid (e, q) \in W\} \text{ and } q^\bullet = \{e \in \mathbf{E} \mid (q, e) \in W\},$$

belong to distinct component ON_i 's; and moreover, $|\bullet q| = 1$ and $|q^\bullet| \leq 1$.

3. The relation

$$(\sqsubset \cup \prec)^* \circ \prec \circ (\prec \cup \sqsubset)^* \tag{2.1}$$

over \mathbf{E} is irreflexive, where:

- $e \prec f$ if there is $c \in \mathbf{C}$ with $c \in e^\bullet \cap \bullet f$;
- $e \sqsubset f$ if there is $q \in Q$ with $q \in e^\bullet \cap \bullet f$.

In Definition 2.3.3(2.1), we use the relation \sqsubset (*weak causality*) to represent a/synchronous communication between two events (see [40, 27]). Intuitively, the original causality relation \prec represents the ‘earlier than’ relationship of the events, and \sqsubset

represents the ‘not later than’ relationship. The input and output sets of a node in a CSON are also extended to include channel places with the relation W . In order to ensure that the resulting causal dependencies remain consistent, in Definition 2.3.3(2.1) we require more than the acyclicity of each component occurrence net, by stipulating that in the graph of the relation $\sqsubseteq \cup \prec$ there is no cycle involving \prec . In other words, the only cycles which can be present are cycles involving \sqsubseteq but not \prec .

The initial marking M_0^{CSON} of a CSON is the union of $M_0^{ON_1}, \dots, M_0^{ON_k}$ (assuming that there are no channel places in M_0^{CSON}). The final marking M_{Fin}^{CSON} of a CSON is the union of $M_{Fin}^{ON_1}, \dots, M_{Fin}^{ON_k}$. In general, a marking in a CSON is a set of conditions and channel places. A step in a CSON is the union of a set of steps which may come from one or more component occurrence nets [45, 60].

Definition 2.3.4 (CSON firing rule) *Let CSON be a communication structured occurrence net as in Definition 2.3.3, M be a marking, and U be a step of the CSON.*

1. U is CSON-enabled at M if $(\bullet U \setminus U\bullet) \subseteq M$.
2. If U is CSON-enabled at M , then U can be fired and produce a new marking M' given by: $M' = (M \cup U\bullet) \setminus \bullet U$. This is denoted by $M[U]_{\text{CSON}} M'$.

The *step sequences* and *reachable markings* of CSON are then defined similarly as for an occurrence net.

The firing rule above means that a step U involving synchronous behaviour can use not only the tokens that are already available in channel places at marking M , but also can use the tokens deposited there by events from step U during the execution of U . In this way, events from step U can ‘help’ each other individually and synchronously pass resources (tokens) among themselves. Thus, in contrast to the step sequence of an occurrence net, where a step consists of a number of enabled events, the execution of a step in a CSON (i.e. $M[U]M'$) may involve synchronous communications, where events execute simultaneously and behave as a transaction. Such a mode of execution

provides possibility to execute multiple events in a single step, and therefore is more expressive than that used in ONs.

2.3.3 Behavioural Structured Occurrence Nets (BSONs)

Behavioural Structured Occurrence Nets (BSONs) are used [42] to model the activities of an evolving system. There are two designated levels which represent an execution history: the lower level for representing behavioural details, and the upper level, which represents the different stages of evolution [60]. Thus, a BSON provides information about the evolution of an individual system. The overall activity phases are used to represent each successive stage of the evolution of this system [42]. For example, as illustrated in Figure 2.4, one level can represent what has occurred in terms of the specific system and its evolution, while the other reveals the behaviour of the systems. In other words, it is correct to state that the former can be viewed as the behavioural abstraction of the latter [42].

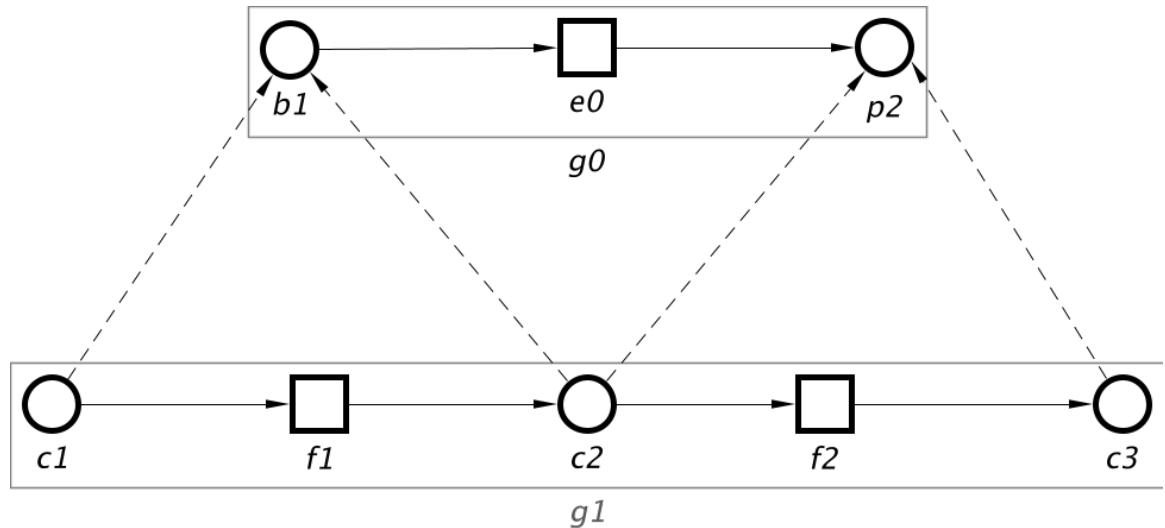


Figure 2.4: Behavioural Structured Occurrence Nets.

2.3.3.1 Synchronous Cycles and Asynchronous Cycles

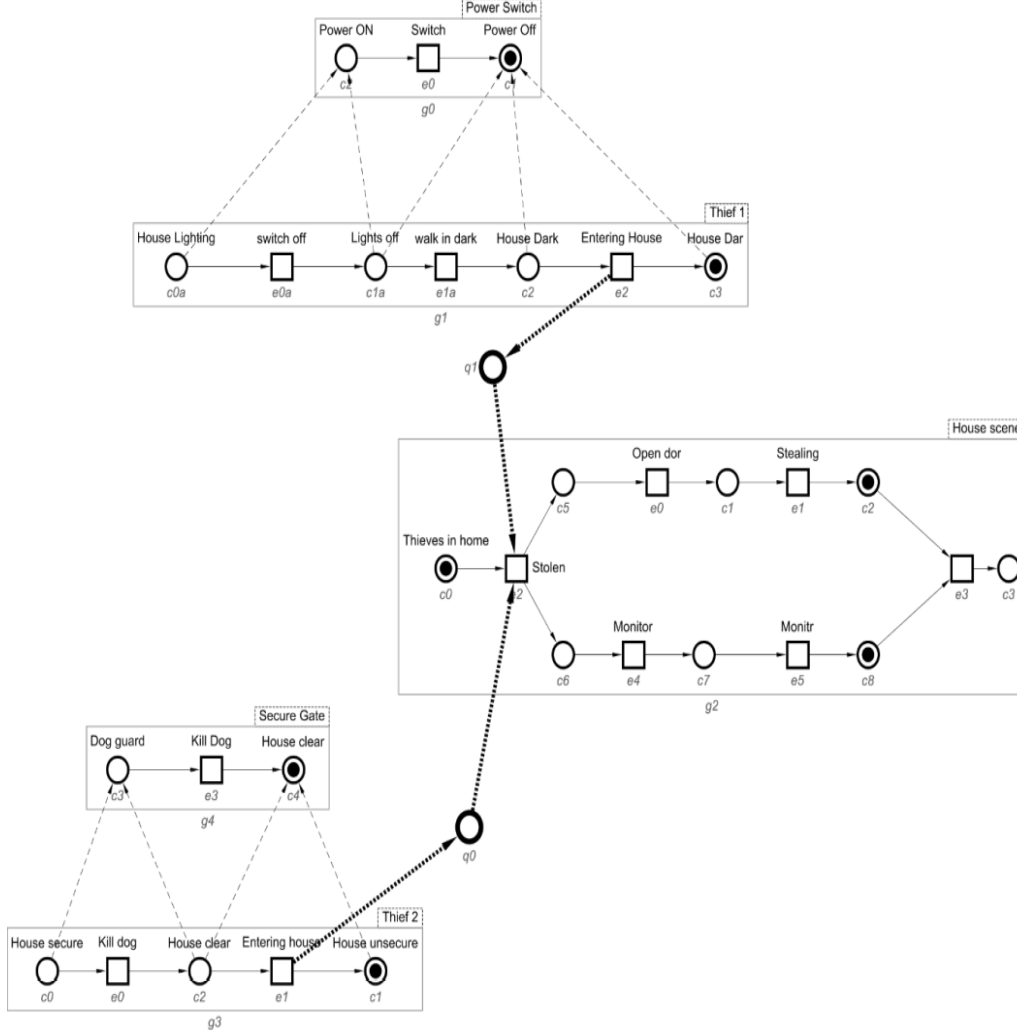


Figure 2.5: Using SONs features in a crime scenario.

In the next example of a crime case, the asynchronous and synchronous communication are used to show how these features help us to model a complicated example. As presented in Figure 2.5, the SON framework can be applied on crime scenarios. The figure illustrates how SON can be used in order to visualise different levels within crime scenarios. This particular model presents five occurrence nets or ONs. Each ON constitutes the behaviour within a sub-scenario. For instance, the individual behaviour of two thieves is represented by three ONs: for Thief 1, Thief 2 and the

House scene. The first ON represents how Thief 1 switched off the power of the house in order to steal from within. The second ON illustrates how Thief 2 cleared the gate by killing the guard dog. Finally, the House scene illustrates the crime scene after the thieves cut the power and cleared the gate. Behavioural relationships are used to hold dependencies between “Power Switch” and “Thief 1”, also between “Secure Gate” and “Thief 2”. Thus, there is a clear asynchronous interaction between two different ONs, i.e. Thief 1 and the House scene, as well as between Thief 2 and the House scene. This interaction illustrates causal relationships between the three different scenarios. As we can see in Figure 2.5 there are several, such as communication between ON(Thief 1) and ON(House scene) as well as synchronous communication between ON(House scene) and ON(Thief 2).

2.3.4 Temporal Structured Occurrence Nets

Atomicity has long been recognised as important concept in distributed and concurrent systems. Many research projects focused on this concept from the perspective of both theory and practice. For instance, [5] models atomicity in asynchronous systems, and [62] focused on how to use atomicity in terms of designing applications. A reasonable way of using atomicity usually leads to reducing the complexity of structured system behaviours. This happens when the detailed system description is hidden, and more abstract view is represented at an upper level simply and clearly. As a result, this can help in the efficient analysis of particular behaviour. For instance, if there is a complicated problem that we need to represent, then the upper level will have simple details, and these details can be abstraction of extensive details at the lower level.

Temporal SONs (TSONs) use the notion of a block to ‘abbreviate’ parts of occurrence net representing atomic actions. Figure 2.6(a) represents the abbreviation of a system using the upper ON, so that the low level view of the behaviour is hidden by

the temporal abstraction [42, 45]. Figure 2.6(b,c) show alternative representations of Figure 2.6(a), where the collapsed behaviour part is replaced by simple events [42, 45].

Figure 2.6: Temporal Structured Occurrence Net.

2.3.5 Time in SONs

In order to analyse and investigate crime scenarios in effective way, the order of event occurrences is very important [46]. Moreover, to determine causes and effects of events it is very useful to obtain their durations. Time in SONs may help to detect suspects and estimate the time of a crime event. Timed structured occurrence nets (Timed SONs) [46] can be used in the analyses and modelling of causality by relating events that have uncertain timing information [46]. Timed SONs can be used in various cases such as crime and accident investigations [6].

For example, the global discrete time can support consistent casual reasoning of a particular system. Moreover, time intervals can be used to capture missed time information. In [46], each node of a SON has start time T_s and finish time T_f . Also, for each time value there is a bounded uncertainty. They are represented by a specified time interval: $[T_{s,l}, T_{s,u}]$ and $[T_{f,l}, T_{f,u}]$, respectively. In [46] each node (i.e. condition, event or channel place) has a duration D with bounded uncertainty represented by a specified duration interval $[D_l, D_u]$, as shown in Figure 2.7.

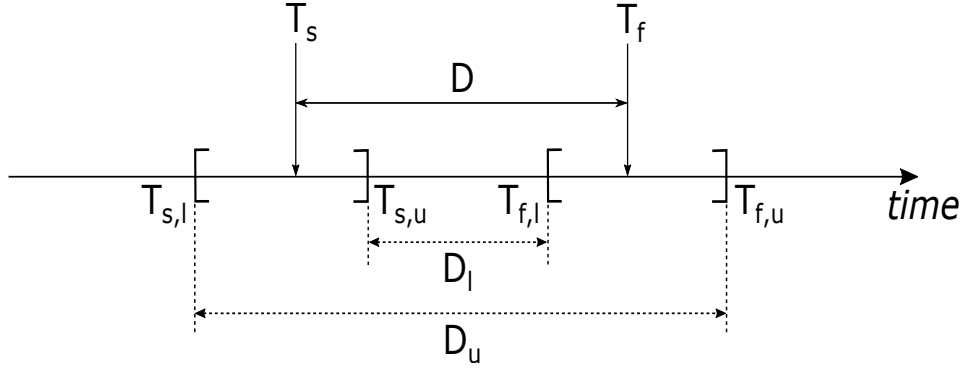


Figure 2.7: Time in SON.

2.4 SONS and Alternative Modelling Techniques

2.4.1 Attack Graphs

Attack Graphs provide a graphical representation of scenarios of attack. In graph theory, a tree is an acyclic graph that connects any two vertices through one path. Also, in general, the visual representations of a decision tree, an event tree, an attack tree, and an occurrence net use an acyclic graph. Attack graphs are visualised and represented through both cyclic and acyclic graphs. Attack Graphs were introduced in 1998 by Phillips and Swiler [54]. After that, attack graphs became a popular representation for attack analysis, starting with [25]. They can be used to detect attacks and defend against them, and also perform forensic analysis. Moreover, attack graphs can be used to analyse the effectiveness of offline detection systems [25]. Attack graphs have been used to model, visualise, and analyse sequences of events of cyber-attacks in computers, networks, and cyberspace. Consequently, they can help the decision-maker to identify security problem in systems [44]. However, attack graphs and attack trees suffer from a distinct lack of standards, prescriptive methodologies, and com-

mon approaches in terms of visualisation [44]. Also, although properly representing preconditions to identify cyber-attacks in terms of mitigation and countermeasures is essential, there is a lack of attack graphs that provide such preconditions [44]. There are several types of attack graphs, for example, the generic attack graphs, the alert correlation graphs, the vulnerability graphs, the miscellaneous attack graphs, and the dependency graphs.

2.4.2 SON vs ML vs Attack Graph

In this section, we will discuss the differences between SON and machine learning in terms of dealing with big data. Before diving deeper into this topic, we first need to define machine learning. After that, we focus on presenting the challenges of dealing with big data in machine learning and SONs. The machine learning (ML) algorithms and techniques have been developed to extract useful and meaningful information from different sources by using training and validating labeled data sets. There are several challenges and issues when ML deals with big data sets. The major problem [65] is that, when the ML technique is trained on a particular data set or data domain, then it is only suitable for this data set or domain; however, it may not be suitable for solving other big data sets. Another issue with an ML technique is that it usually uses a different number of class types, by which we mean a larger variety of class types that are found in the dynamic growing data set which causes inaccurate classification results [65]. Furthermore, an additional challenge is that ML techniques have been developed to rely on learning one task at a time. As a result, these techniques are not suitable for multitask learning and knowledge transfer that today's Big Data analytic require. Moreover, dealing with big data in real-time scenarios is still a challenge in ML [65], because dealing with such situations requires constant user interaction. On the other hand, SONs can deal with large data set using features like causality and concurrency. However, SONs are still in their

infancy when it comes to dealing with large data sets. We have investigated this problem and provided a SON algorithm dealing with large data sets. However, SONs need to be improved in terms of dealing with big data. For instance, research on SONs should aim to improve the visualisation of big data, by combining it with other technologies such as Hadoop, Neo4j, and NLP (Natural Language Processing).

This comparison leads to an important conclusion that a major difference between SONs, ML, Attack Graphs, and general Petri nets is that the structuring and abstraction mechanisms in SONs (such as the behavioural abstraction) are not supported by the other frameworks. In addition, Attack Graphs and (in particular) ML do not provide an explicit representation of causality and concurrency as SONs do.

2.5 Visualisation

In this section we will focus on the background of visualisation and its importance. In addition, it will review visualisation methods and techniques.

The goal of visualisation, as defined by [64, 39, 36, 72], is to help the researcher or the decision maker to easily identify insights which result from data. Visualisation helps to present the respective data in a suitable and understandable manner. For [68], visualisation represents the best way to convey complicated ideas clearly, precisely and efficiently. These kind of graphical representations are understandable and easy to explain. The aim of visualisation is to generate useful insight from large amounts of data. In addition, visualisation also helps to analyse, discover and clarify information while placing it into an understandable form. The main purpose of using visualisation is given by the need to synthesize and present a large amount of data, often from different sources and viewpoints, with the purpose of describing different levels of detail [50]. Visualisation relies on data discovery methods which help users to combine different data sources to create new analytical views.

2.5.1 Importance of visualisation

Discovering relevant meaning from figures, images and presentations is generally easier than from other forms of information [68]. As such, visualisation is not only a way to provide information and present it, but also a suitable means of communication [50]. Today, visualisation can help in different ways. For instance, high velocity, enhanced optimisation and significant impact lead to insight discovery [38]. All these points are ultimately used as a support by both decision makers and researchers. Visualisation data [15] is a main key of the discovery process when operating with big data sets. Moreover, visualisation is important for extracting knowledge and understanding and recognising interesting patterns. It also serves to identify correlations and outliers. As a result, visualisation is able to provide a plethora of techniques which can help to understand and analyse evolving data.

2.5.2 Visualisation stages

Visualisation occurs in four stages [69], as illustrated in Figure 2.8 below:

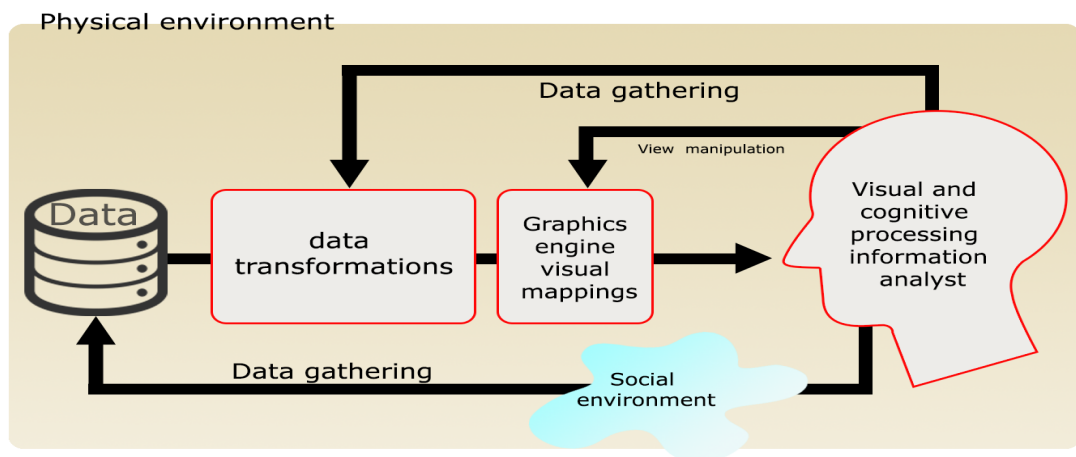


Figure 2.8: Visualisation process [69].

Step 1: Data collection and storage.

Step 2: Preprocessing: in this stage, data is transformed to be easy to operate and understand.

Step 3: Using algorithms which produce and present data into the computer as an image.

Step 4: The human perceptual and cognitive system [69].

2.5.3 Visualisation techniques

Considering the importance of visualisation today, in most cases, simply presenting information is not enough. During the last decades, various data visualisation techniques have been elaborated. As a result, visualisation achieved a considerable level of development. This makes it possible to deal with huge amounts of data in an efficient manner. In addition, new, different methods of data visualisation appear and are being developed constantly. These different visualisation techniques are helpful to conduct research in information science. Crime investigators and decision makers in government also rely on it to gain insights. Moreover, it is important to harness advantages such as human cognitive capacities in order to achieve even more accurate results [47]. This section reviews several of the most important visualisation techniques. In addition, the well-known principle of usability is brought into the discussion.

Types of interaction techniques

Visualisation techniques [47] are classified into the following two main categories:

1. Technical Approach (data structures oriented).
2. Interactive approach (user oriented).

The following section describes these two different types of visualisation. Each is defined and illustrated with examples.

2.5.3.1 Technical Approach

This subsection focuses on different data structuring, the first of which are linear data exploits. Several visualisation techniques are included here:

- a) Timeline technique. This technique allows users to follow temporal events from different points of view. It can be defined as a graphical representation which helps to illustrate a given list of events in the chronological order in which they occurred. This technique presents each object as an event line which shows the start time and end time of each event [56]. Moreover, the values which are attributed on the timeline illustrate how they changed over time. In addition, it supports queries and filtering [23]. The timeline technique allows users to easily recognise whether there are any particular patterns which could be shown in any particular time. Also, the technique helps to illustrate how different events can be distributed during different time periods.
- b) Using data tables which allow users to visualise a large set of data in a table form [55]. This technique provides an overview, filters details and can easily recognise if there are any relationships between objectives. However, this technique presents a challenge, namely it cannot extract the valuable data only by identifying it [47].

The second technique is multidimensional data. This technique has been used to reduce the number of dimensions to two or three, with the purpose of making it easier to translate and analyse the data. There are two types of multidimensional data techniques:

1. Scatterplots which showcase data through groups of dots [19]. It is suitable for filtering information. However, the multi points cause difficulty when reading the data [47].

2. The polyline system, which is used to represent each object of the base by polyline with a number of data dimensions within the system [47]. Data dimensions are represented in each axis. The most well-known techniques are called parallel coordinates [26]. These techniques allow the user to visualise many dimensions at same time [47]. Through it, model tendencies and correlations can be established. This type of visualisation is useful to discover any phenomenon which is otherwise hard to detect [47]. However, the main lacking of this technique is that it is important to exploit all parameters in order to detect all views. The third data structuring technique is that of hierarchies. In this technique, data is hierarchically organised through a tree structure which links parent and child nodes such as graphs and trees. This technique is suitable for information visualisation because it is clear and simple for users. Moreover, when compared with other visualisation techniques, it provides the benefit of swift testing and diversity [61]. There are several different types of hierarchy techniques:
 - a) Metrics-wise [20]. It includes Boolean metrics which have false values by default. In each tuple in the graph, the line and the column which are intersected both have true value [20] shows that this technique can detect the underlying structure of the graph. Moreover, we can use this technique to identify models in space. The technique is also useful for data mining [47]. However, through this technique it is difficult to visualise many binary relations within one matrix. In this technique, data is set in a list.
 - b) The Tree Map is used to represent hierarchical data within a form. In other words, the technique is used to represent hierarchical structures [29]. It allows the user to easily compare different nodes at different depths. Also, it helps to redefine and find new patterns.
 - c) The Cone Tree is another type of technique which helps to visualise hierarchical

data in three dimensions. It is named as such because its branches expand in the form of a cone [8]. It also possesses an attractive overview. The base consists of notes which are the top of the cone. In addition, sub-cones are formed around the base, in a circular fashion. It can take either a vertical or horizontal form [8].

The fourth technical approach technique consists of Networks. It is a graphical visualisation of the logical relationship between different aspects. It generates a direct graph and combines nodes, edges and arcs. In each arc there is a label. The nodes represent concepts, and the arcs represent relationships.

2.5.3.2 Interaction techniques

Any system which visualises information has two main components, namely representation and interaction [71]. The representation component is situated in the field of computer graphics, which is responsible to for the mapping process, from data to representation. However, the other component is interaction, which is accountable for the interchange of information between users and systems, so that the users can discover varieties of the overall data set to get insights [47]. Moreover, interaction components are found in human computer interaction (HCI). It is a challenge to provide a clear definition of interaction in visualisation, defined as “the communication between users and the system” [12]. However, interaction techniques are easier to define and indeed a more tangible concept than interaction. Interaction techniques are designed to change and adjust visual representation instead of just entering data into systems [12]. Moreover [71], interaction techniques act as features which allow the user to translate and manipulate representations either directly or indirectly. Also, interaction techniques can improve user perception of information during working with a particular data sets [22]. The main advantage of interaction is that it offers the ability to interact with visual representations. In so doing, it can deal with overlapping objects and visual mess, especially when dealing with large data sets. As a

result, it is a technique which is able to handle complexity in such issues, namely overlapping objects and visual mess [12]. These techniques had been developed by Keim [36], who classified them into several main categories.

1. Distortion Techniques

This technique helps the process of data exploration by focusing on details while preserving the general data overview [37]. The main idea of distortion techniques is that they display part of the data with high level of details, while at the same time displaying the parts which have a low level of details. In [36], Keim distinguishes between dynamic and interactive distortion techniques, which ultimately change if visualisation is made automatically or manually.

2. Dynamic Projections

The main idea of this technique is that the change to projections are made dynamically, with the purpose of discovering multidimensional data sets [37]. The disadvantage of this kind of technique is that it is difficult to operate data sets.

3. Interactive Filtering

This type of technique is suitable for large data sets because it divides the respective data sets into segments, making it easier to focus on the interesting sets [37]. This process can be done by directly browsing each particular subset, or by specifying the properties of each particular subset. However, browsing is difficult when operating with large data sets and also querying may not provide the desired effect [36]. As a result, there are several interaction techniques which have improved the interactive filtering to boost data exploration. For example, Magic Lenses which use a magnifying glass tool which supports data filtering within the visualisation. Moreover, data operate under the magnifying glass tool and can be filtered, meaning that the resulting data will be represented

differently [36]. Magic Lenses shows the modified data view which the user has selected, while the general visualisation remains unaffected. Different lenses with different types of filters are available and can be used in Magic Lenses.

4. Interactive Zooming

The zooming technique has been widely used in practice and is a well-known technique. It is typically used when operating with large data sets. Furthermore, it allows data to be visualised as compressed, to display the data overview [36]. However, it also allows data to be displayed in different resolutions. Additionally, although it is used to show the data in a large view, it is also used to automatically visualise any changes to the data. Furthermore, it allows that view to be expanded, so as to allow the user to view more details on a high level zoom. For example, objects are displayed as pixels within a low level zoom, as icons on an intermediate zoom level, and as labelled objects at a high level.

5. Interactive Linking and Brushing

In this technique, different visualisation methods have been combined to make up for the disadvantages of other visualisation techniques. For example, scatterplots of different projections can be combined by colouring and linking each subset of points and all projections [36]. Moreover, connecting different visualisation techniques by interactive linking and brushing provides more details on independent information [36].

2.6 Domain Name System (DNS)

A DNS or Domain Name System assigns domain names and then maps each of them in accordance with the existing name servers, which host each respective domain [51]. At the same time, due to various reasons, network administrators are allowed to delegate

authority over some sub-domains of a given name space to other name servers. This permission was given so that the existence of a single, large, centralised database would not occur; at the same time, the resulting services are fault-tolerant [51]. Additionally, DNS also defines the technical parameters of each database service which is situated at its core. These parameters are part of what is known as the DNS protocol, meaning a detailed specification of both the data structures within the DNS, as well as communication between these various structures. Both these elements make up the Internet Protocol Suite. As a broad rule, two main name spaces are maintained across the internet, namely name hierarchy [51] and IP or Internet Protocol address spaces. DNS accomplishes two tasks: on one hand, it ensures the domain name hierarchy is kept, while providing translation services between it and a given address space, on the other hand. To function properly, DNS relies on multiple servers as well as on a communication protocol. Records for a specific domain are stored on a particular type of DNS server called a DNS name server; it answers to queries addressed to databases within it. Once a user types in any web address in the browser, for instance `www.google.com`, the current, local DNS server (resolver) will check whether or not the address is stored within it and then sends a response to the user, with the IP address of that particular website. However, if the resolver does not recognise that domain, it will send the query to the DNS root name server. As a result, the DNS root server will send a response back to the resolver, telling it the domain name (`google.com`) belongs to name server for `.com` top-level domain (LTD). Afterwards, the resolver will send the query to the DNS server `google.com`, asking about the IP address of `www.google.com`. As a result, the `.com` name server will respond with an IP address for google and the resolver will receive it and open the website for the user [51]. Figure 2.9 illustrates this process.

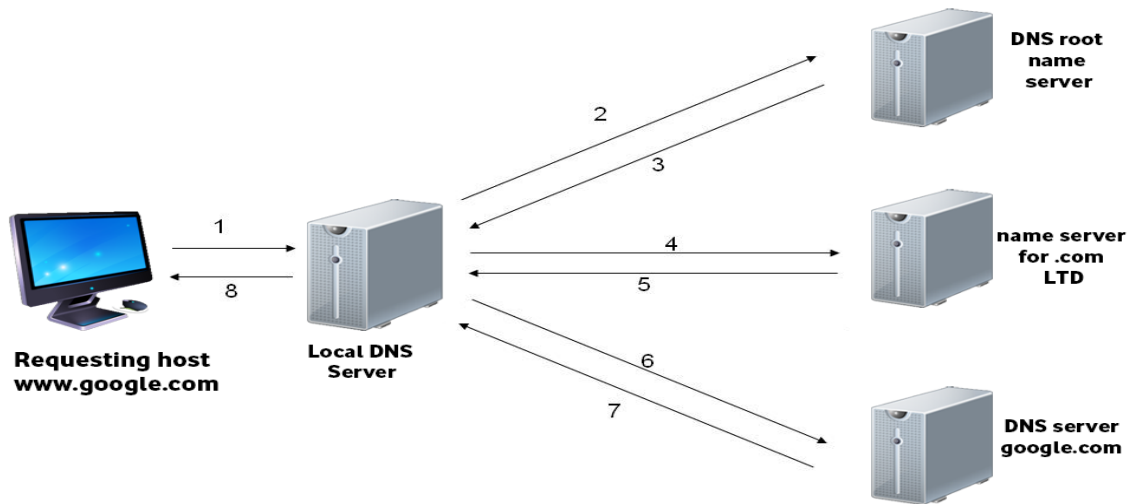


Figure 2.9: How DNS Resolves Names and IP Addresses (the labels on arcs indicate successive steps of DNS protocol).

2.6.1 Name servers

The maintenance of Domain Name Systems is ensured through distributed database systems, which use client - server models. In these cases, database nodes consist of name servers. At the same time, for each existing domain, one or more given DNS servers provide data concerning the respective domain and the servers of domains which are subsequent to that respective domain. At the top of this hierarchy, we find root name servers, which typically receive queries when a LTD must be solved [30, 34].

2.6.2 Location transparency

Location transparency helps to identify existing an network resources, through names, rather than through location [51, 30]. For instance, a user gains access through a unique file; however, actual data is reserved in sectors which are dispersed either across the network or within a local computer. Within such a system, the actual location of a file is not relevant to the user; however, a distributed system is required, so as to ensure a naming scheme for the available resources. One of the essential

gains of using location transparency is that location is indeed irrelevant. Considering the given network setup, a user can obtain files from any computer connected to the network. As such, the actual location of a given resource does not matter anymore, thus creating the impression that the whole ensemble is accessible from any computer terminal, further boosting software upgrades [51]. Using location transparency also provides considerable versatility. This means system resources can be shifted from one computer to another, within the network, without disrupting the network. Simply updating the location of a given resource is enough, as all programs using that specific resource will be able to find it [51]. In a nutshell, location transparency is highly user friendly, as data is accessible to everyone who connects to the internet and who knows the name of the file or files and who has the required clearance to access it.

2.6.3 DNS records

The DNS protocol typically uses multiple record types for various purposes. The most common type of such records is A, which essentially switches domain names into IP addresses. For example, AAAA records accomplish a similar role for IPv6 protocols. Other record types, such as CNAME, are known as aliases. This type of record directs towards another defined domain name, but never towards an IP address. At the same time, one IP address can have several domain names or aliases. NS stands for NameServer; it keeps the address of the authoritative NameServer which can resolve any sub-domains for that.

2.6.4 DNS Tunnelling

DNS tunnelling is a covert communication channel, which allows encapsulating the traffic of other protocols (E.g.: HTTP, Telnet, FTP, SSH, etc.) within DNS packets. This results in concealing the real protocol by making the traffic look like ordinary DNS traffic. DNS tunnelling is very suitable to be used for malicious activities such

as data exfiltration as well as command and control call-backs from within restricted internal networks. Different from other tunnelling methods such as SSH Tunnelling, a distinctive property of DNS Tunnelling is that, it uses the legitimate DNS servers of the internal network to hop its packet through, to reach the final destination, which is the attacker-controlled server. DNS Tunnelling encapsulates the data in DNS queries and responses. In this way, the outer protocol remains as DNS; however, the encapsulated data payload belongs to another protocol [30].

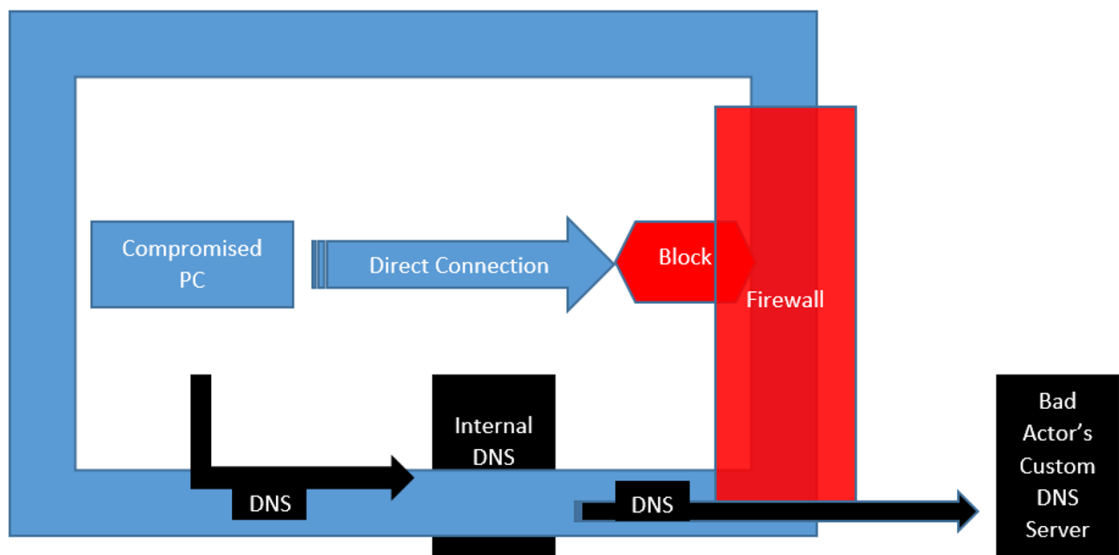


Figure 2.10: DNS tunnelling.

Figure 2.10 explains how DNS tunnelling is performed. The powerful advantage of DNS tunnelling is that, if a compromised computer is trying to send “secret” data to the attacker-controlled server on the external network (the Internet), even if such direct communications get blocked by the network firewall, DNS Tunnelling would bypass this restriction. The compromised computer would send the secret data within a Base64-encoded message, for example “c2VJcmV0”. It will send the query to the internal DNS server, to ask the corresponding IP address, in the form of a basic DNS query for c2VJcmV0.talalncl.com. The internal DNS server would forward this DNS query to the authoritative DNS server for the DNS Zone “talalncl.com”. Since the authoritative DNS server is actually the attacker-controlled external server, the

message gets delivered to its destination. The talalnl.com DNS server will then respond and return a dummy answer, such as 127.0.0.1. Afterwards, the talalnl.com DNS server will analyse the query as follows:

c2VjcmV0.talalnl.com

Base64Decode

c2VjcmV0 – > secret which means the data extracted from within the DNS query.

Finally, a single message within DNS tunnelling looks normal. This message may look exactly like a normal DNS query. This is actually the weakness of DNS Tunnelling: although a single packet may look like a normal DNS query, thousands of queries for sub-domains within the same domain will help to detect the DNS tunnel.

2.6.5 Existing Detect DNS Tunnelling

There are several techniques that can be used to detect DNS tunnelling. The next table shows that.

2.6.5.1 Traffic Analysis

1. The volume of DNS traffic per IP address

One way to detect DNS tunnelling using traffic analysis methods is to look at the traffic that a specific IP address generates (Pietraszek, 2004). The technique works based on the fact that each request of tunnelled data is up to 512 bytes, therefore, a complete communication will require one to send many requests (Van Horenbeeck, 2006). A server will send the request continuously if the client is polling the server.

2. The volume of DNS traffic per domain

The other fundamental method of detecting DNS tunnelling involves looking at

the amount of traffic to a particular domain. Because utilities are designed to tunnel data through a specific domain name, this method can be used to detect possibility of DNS tunnelling (Butler, 2011). However, if DNS tunnelling is configured with multiple domain names, each domain receives less amount of traffic.

3. Number of Hostnames per Domain

Guy (2009) suggests that number of hostnames per domains may indicate possibility of DNS tunnelling. In DNS tunnelling utilities each request has a unique hostname, which means that there will be a large number of hostnames compared to a legitimate domain name. However, this method can be sophisticated to determine an optimal threshold since various methods of tunnelling varies when it comes how many hostnames it has.

4. The DNS Server Geographic Location

Another factor to consider when determining DNS tunnelling is where the DNS server is located geographically. If a given location where a DNS server is receives or sends significant amount of traffic, this may be an indication that there is DNS tunnelling in that part or parts of the world (Skoudis, 2012). This method may be of help for organizations that do not have businesses in the international markets.

5. Domain history

Domain history can help to detect DNS tunnelling, as it can raise suspicion on DNS traffic. Domain history is used to detect tunnelling by determining when A or NS records were added since a domain may have been acquired not long ago for DNS tunnelling, meaning that its NS was added recently (Zrdnja, 2007). The method can be used to detect the domain that has been involved in malicious activities.

6. Volume of NXDomain responses

DGA generated names can be detected by looking for excessive NXDomain responses (Antonakakis, 2012). The volume of NXDomain can be useful when one needs to detect Heyoka, which is capable of generating a large amount of NXDomain responses.

7. Visualisation

Visualisation may be used to detect DNS tunnelling (Guy, 2009). Visualization involves interactive work done by an analyst; however, tunnelled traffic is noticeable.

8. Orphan DNS requests

Most of the methods used to detect DNS tunnelling look for visible evidence of tunnelling, however, other approaches may look for what is missing or what one expects to see. When it comes to general computing, a DNS request follows another request done previously, for instance, a DNS request for a web page will follow HTTP request. Orphan DNS requests lack consistent requests coming from a different application for instance http and some exceptions will be easily filtered out. Some security devices store the process of retrieving information by a given IP address while DNS requests are used in anti-spam solutions to locate backlisted IP addresses. To check the reputation of the suspicious file, endpoint security makes use of DNS requests with encrypted file hash that is rooted in the fully qualified domain name (FQDN).

9. The general covert channel detection

Research has addressed various techniques for covert channel detection that are not associated with the protocol (Couture, 2010). One may use utilities set up to detect tunnel by looking at things such as request time or even to compare traffic to a statistical fingerprint.

2.6.5.2 Payload Analysis

1. Request and Response Size

Analysing the size of the request and response is one payload analysis technique used to detect tunnelling. According to Bianco (2006), the source byte to destination byte ratio can be used as a method of identifying suspicious DNS tunnelling traffic. MySQL database stored data as part of Squall or Snort systems is requested to determine the destination and source bytes. Then, one compares the source byte to destination byte ratio with the limit value. Because DNS tunnelling unities attempt to feed as much data into the request as possible, it is possible that tunnelling request will contain ling labels up to 63 characters as well as ling overall names which are up to 255 characters long. Looking for hostname requests that are longer than 52 characters may be another method of detecting DNS tunnelling (Guy, 2009).

2. **Entropy of hostnames** DNS names that are genuine are made up of dictionary words or something else that looks meaningful. However, DNS names that are not legitimate are unpredictable and tend to use more character sets. However, cases where DNS names represent some type of information, for instance, content delivery networks, can be excluded. Therefore, DNS names with high entropy are an indicator of DNS tunnelling.

3. **Statistical Analysis** Another technique of determining DNS tunnelling is to observe the frequency at which specific characters in DNS name occur. While genuine DNS names have few numbers, encoded DNS names tend to contain a lot of numbers. Another proposed methodology is to look at the proportion of specific letterings in the domain name (Bilge, 2011). Another character based method of detecting tunnelling is to consider the proportion of the Longest Meaningful Substring (LMS) method (Bilge, 2011). One can also look at the

number of unique characters, and alert on anything that has over 27 unique characters (Guy, 2009). It is possible to detect names generated using DNS tunnelling considering the fact that genuine names mirror common language to a given degree. Another statistical analysis method of detecting DNS tunnelling is to look for repeated consonant and numbers. It is common for tunnelling utility to create names with consecutive number and consonants unlikely to be seen in domain names mirroring common languages (Lockington, 2012).

4. **Uncommon Record Types** This method involves analysing record to find those that typical clients do not use regularly. These uncommonly used record types can be used to detect DNS tunnelling. Record types such as TXT are rarely used (Pietraszek, 2004), and if one identifies them, he or she can detect tunnelling of DNS.
5. **Policy Violation** In some cases, all DNS requires a specific policy just like it all DNS is required to pass through an internal DNS server. Therefore, in case there is a violation of this policy, it will be a good indicator of DNS tunnelling. This method requires monitoring the DNS requests directly to the internet (Fry, 2009). It is important to note that most of DNS tunnelling utilities are designed in a way that they can work even as they forward request through a DNS server that is internal.
6. **Specific Signatures** Researchers and authors recommended the use of a particular signature of the DNS tunnelling names. Most of the DNS tools have DNS names with particular signature or patterns. A signature can help on checking particular attributed in a DNS header as well as checking for particular content in the payload. For instance, there is a methodology that has been recommended for detecting DNS tunnelling and it is called Snort signature (Van Horenbeeck, 2006).

2.7 Conclusion

In this chapter, different background areas have been reviewed and discussed. Also, the motivation behind this project has presented. Moreover, an overview of structured occurrence nets (SONs) and its features have presented. Also, we have provided the SON features with its definitions. For instance, communication in SON (CSON) and provide examples that show how these communications occurred. Moreover, we have discussed the domain name system (DNS) and review the methods of detection that DNS tunnelling. After reviewing and investigating this matter, we have come with next result. During the review of the existing DNS tunnelling detection, there were many detections methods. Many rely on traffic analysis, such as the volume of DNS traffic per IP address, the volume of DNS traffic per domain, and so on. However, (Guy, 2009) suggests visualisation as a method to detect DNS tunnelling; however, it argues that the tunnelled traffic stands out dramatically. Compare this with our solution in chapter 4, the feature of structured occurrence net allows us to deal with this issue in a proper way. I mean in term of visualising the data in SON can give a away to detect DNS. Other methods to detect the DNS tunnelling rely on statistics. However, our approach is to create a new algorithm that detects DNS tunnelling by modelling DNS packets whether that normal or abnormal packets and our algorithm will distinguish the abnormal and detect it, see section 6 in chapter 4.

Chapter 3

Visualising Data Sets in Structured Occurrence Nets

3.1 Summary

Visualisation techniques have been used with increasing success to assist users in complex system analyses, providing meaningful insights. Visualisation of SONs in tools such as SONCraft is currently limited, as there is a lack of effective support to visualise complex and large data sets. In this chapter, we address challenges resulting from the overlapping and out-of-order placement of ONs, and propose a novel solution to deal with this issue based on a step execution policy. Also, we discuss the development of the resulting algorithm, and SONCraft plug-in implementing it. In this chapter we proceed as follows. The next sections 3.3 and 3.4 discuss visualisation challenges related to SONs in SONCraft. In Section 3.5 we propose a novel solution to deal with them based on maximal concurrency. Section 3.6 describes the policy-driven visualisation algorithm. Section 3.7 describes implementation and results. The last section 3.9 contains concluding remarks.

3.2 Introduction

Complex and large data sets drive considerable attention from information science researchers, crime investigators, and decision makers within governments. Tremendous growth of this kind of data tends to transform its analysis into a hard process. The existence of such data provides important insights, but also raises challenges. In particular, visualisation techniques have been used with increasing success to assist users in various analyses, and providing valuable insights about system structure and behaviour [32, 63].

Although SONs are supported by visualisation in tools such as SONCraft [46, 45] there are currently limitations as one cannot effectively visualise data sets automatically. Our aim is to enhance SONCraft making it possible to visualise big data sets in an helpful and understandable way. The approach we propose is to use *semantically driven visualisation* which adopts maximal concurrency (a step execution policy in the sense of [9]) to provide structured displays. As a result, the new approach should lead to effective display of complex SONs.

3.3 Policy Driven Visualisation

The current section evaluates SONCraft from the perspective of interaction techniques, as there are issues to be expected when inputting data to SONCraft, after allowing the tool to accept data sets automatically. Such issues include the overlapping of ONs. They can be addressed by using the concept of maximal concurrency, which is a step execution policy in the sense of [9]. Concerning the implementation of new visualisation techniques, they deal with the two main purposes of data analysis, namely understanding and exploration. The aim of exploration in data analysis is to find a perspective which has not been previously discovered. A good example is a crime investigator who searches for behavioural patterns of criminal gangs. As a re-

sult, this will allow us to develop SONCraft so that it will be able to help investigators in detecting subtle patterns (modus operandi) embedded inside data automatically retrieved by SONCraft. However, some problems are to be expected, and they are discussed in the next section.

3.4 Loading data sets in SONCraft

Although SONCraft allows visualising data in an attractive way, it does so mainly when cases are manually modelled. The main issue which users face while attempting to load data automatically is overlapping and out-of-order placement. As shown in Figure 3.1, we should expect both problems after inputting data sets into SONCraft.

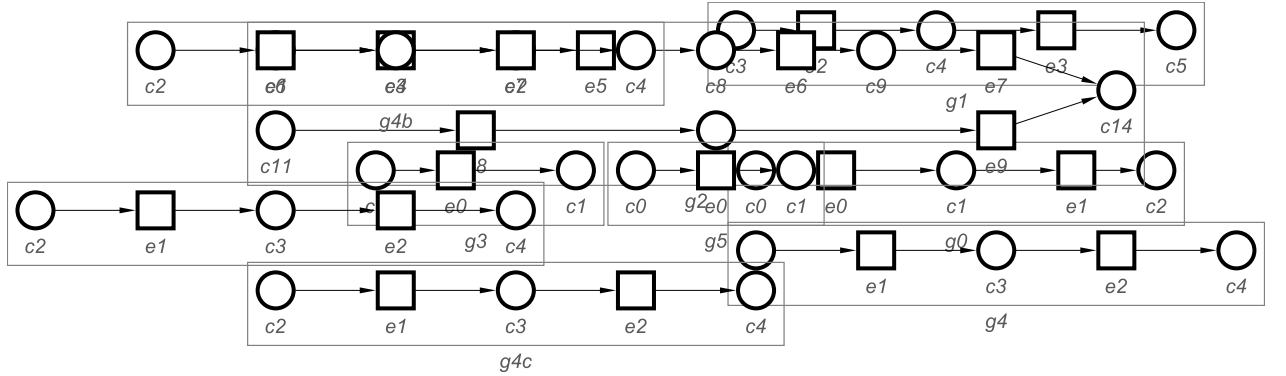


Figure 3.1: Overlapping and out-of-order ONs.

Figure 3.2 shows a better way of presenting the same data which avoids overlapping, but still places ONs in an unstructured way. The latter problem will be addressed by applying the idea of maximal concurrency to order the events, and so also the ONs.

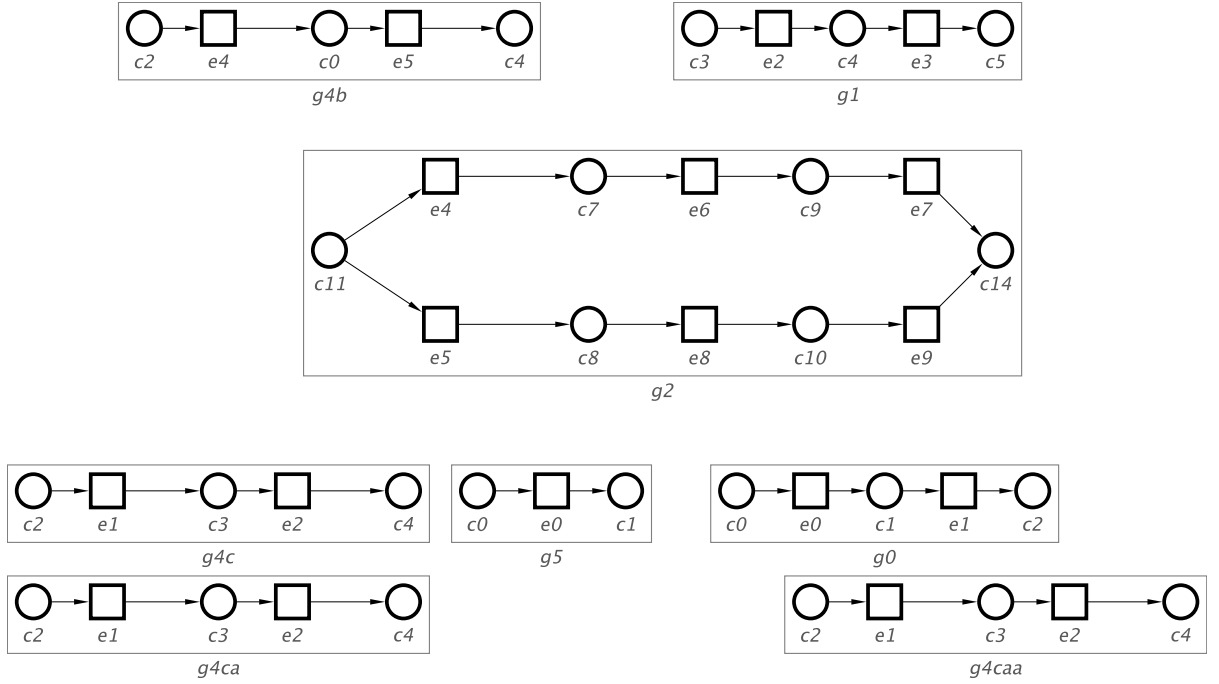


Figure 3.2: Removing overlapping of ONs.

There are various reasons due to which overlapping occurs. The first one is due to the fact that although data is loaded automatically when it is retrieved, it is loaded randomly. The other reason is given by the fact that the current SONCraft platform is not prepared to deal with data in an automatic way.

3.5 Proposed solution

Our approach to solving visualisation problems related to SONs consists in a careful consideration of the positioning of the ONs within the workplace area. The method involves the arrangement of ONs in a particular way so as to prevent and avoid overlapping and out-of-order placement by using maximal concurrency (a step execution policy in the sense of [9]). The novel idea is to allow users to automatically control and retrieve data from SONCraft leading to what one might call a *policy-driven visualisation*. This approach allows the visualisation of concurrency in display systems which are capable of representing it, such as SONCraft.

3.6 Policy-driven visualisation algorithm

We have taken advantage of the well-known algorithms such as topological sorting and Tarjan algorithm. The idea behind the algorithm is explained and explanation is supported by clarifying examples. Our consideration to solve the visualisation challenge led us to use topological sorting which treats a directed graph as a basis of a linear ordering for that graphs vertices [33]. In order to identify sets of maximally concurrent events in SONs, we needed to find executable events and then co-locate all transitions as maximally concurrent subsets in a particular set X .

For example, in Figure 3.3, there is a maximally concurrent set of events comprising event a in ON $g0$ and event f in ON $g1$, so these events are presented as one cluster. After placing a and f in the same maximal ‘slice’, events b , c and g represent a maximally concurrent execution and, similarly, after placing them in the same maximal ‘slice’, events d and e form a maximally concurrent set.

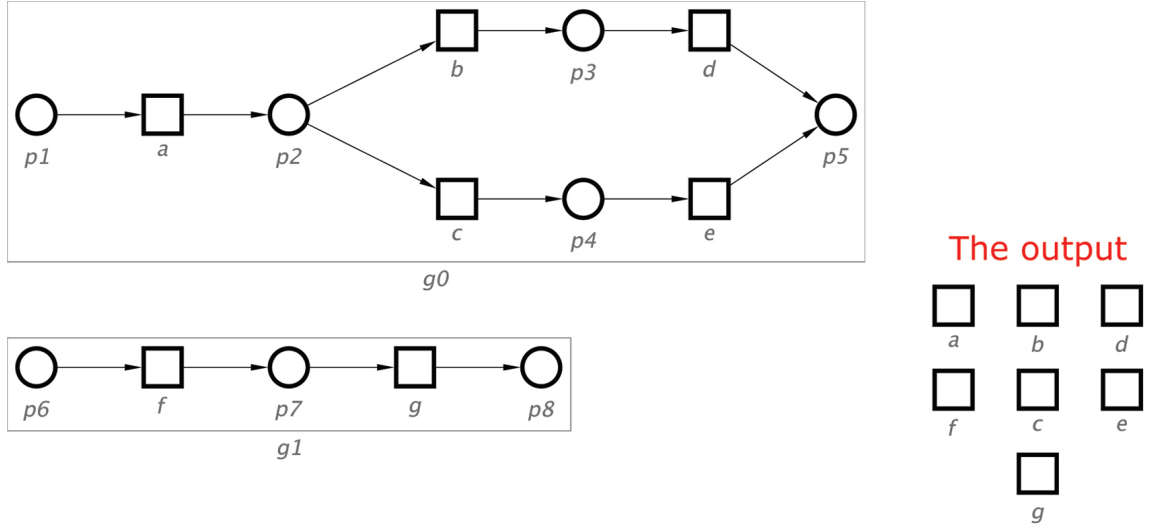


Figure 3.3: Maximally concurrent representation of events.

The result of applying visualisation based on the maximal concurrency policy discussed above is illustrated in Figure 3.3. As shown, all events have been manually clustered into maximally concurrent steps. However, our aim is to allow data sets to

be retrieved automatically rather than manually. As such, several algorithms have been considered to assist us in doing so.

As Khan [33] emphasises, the main reasoning behind topological sort algorithm is to find a list of start nodes which have no incoming edges [33] and then inserting any such node into a particular set X . After that, the node is deleted from the graph and further nodes with no incoming arcs are identified. The development of such an algorithm will help us in finding sets of maximally concurrent events. Thus, our approach consists in using the idea of a topological sort algorithm with the purpose of finding maximal sets of events (steps) which are ready to be executed. In order to identify the right events, we would need to find an event which has no previous events connected to it. Although SONS are semantically acyclic graphs (in the sense of step sequence semantics) and the topological algorithm was working properly, a challenge arose during the process. It was due to a special kind of cycle (weak causality cycle) within SONS, which occurs when two or more different ONs communicate with each other through channel places. Figure 3.4 illustrates this situation.

As a result, initially, our algorithm could not handle such a situation, because if it was to be applied, for instance, in Figure 3.4, then event a had another event connected with it, i.e., event d . Our approach copes with these problems, should they occur, by using Tarjan algorithm. This algorithm was adapted so as to allow one to detect this special kind of cycle (i.e. weak causality cycle).

The adaptation implied the ability to push every event of a particular cycle in one set as a maximal concurrency cluster, should this type of cycle occur. For instance, after applying the algorithm to the graph in Figure 3.4, the first cycle between the three existing ONs, namely ($ON2$, $ON3$ and $ON4$) was detected. This was made via channel places ($q0$ and $q1$) and ($q2$ and $q3$). As such, the algorithm will push all events in this cycle, namely events (a , d and f), to one set. After that, event b is pushed into one set. Finally, the second cycle between $ON1$ and $ON2$ is detected via

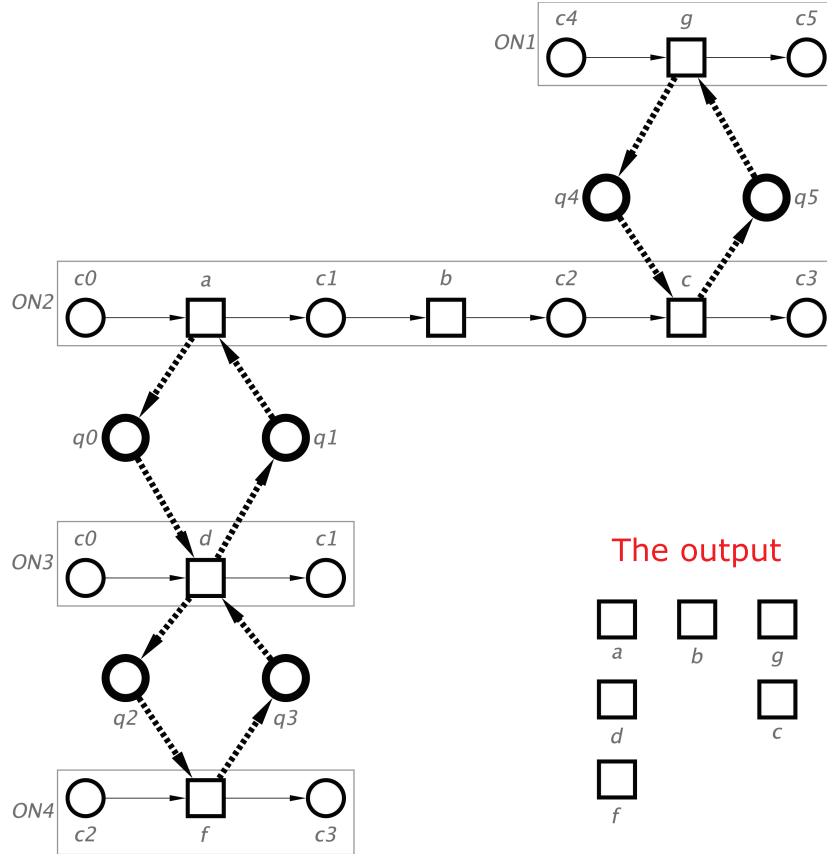


Figure 3.4: Special kind of cycles in SONs.

channel places ($q4$ and $q5$) in one set, namely events g and c . Figure 3.4 illustrates the result. The other interesting case that we have discovered during our investigation of policy-driven visualisation in SONCraft is shown in the next example, depicted in Figure 3.5.

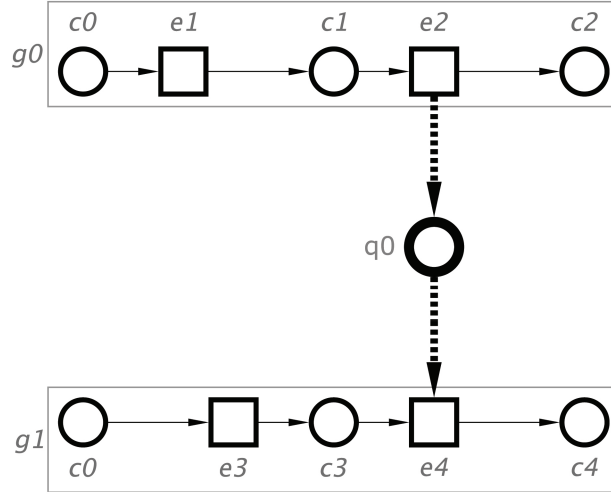


Figure 3.5: The case of asynchronous communication.

In this case, if we apply algorithm based directly on topological sort, we would group events $e1$ and $e3$ as a set, then take event $e2$ as a singleton set, and finally event $e4$ as another singleton set. Figure 3.6 shows the result. Although events $e2$ and $e4$ can occur together after $e1$ and $e3$ due to asynchronous communication, this could not be handled by the standard algorithm, because event $e2$ would be treated before event $e4$. A possible solution is provided by an updated algorithm where, after detecting asynchronous communication, one pushes all events within it into the same set.

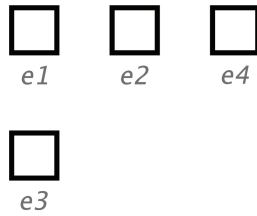


Figure 3.6: The standard Result.

Figure 3.7 shows the result obtained after applying this solution to our example.

ALGORITHM 1: Visualize SONS' events By Maximal Concurrency

Input: set_of_ons : Set of ONs ;

Output: set of maximal sorted ONs' transitions

```
1 Start ;
2 array set_of_ons_to_check = set_of_ons ;
3 array set_of_maximal_concurrencies = array[array[Node]] ;
4 array temp_maximal_concurrencies = [] ;
5 boolean loop_again = true ;
6 while (loop again) do
7   loop_again = false ;
8   array temp_maximal_concurrencies = [] ;
9   for (ONs ons in set_of_ons_to_check) do
10    array ons_nodes = ons's nodes;
11    array ons_set_of_cycle = [] ;
12    ons_set_of_cycle = Tarjan(ons);
13    for (Node node in ons_node) do
14      if (node is a transition) then
15        loop_again = true ;
16        if (node has no previous transition) then
17          boolean node_belongs_to_a_cycle;
18          for (Cycle cycle in ons_set_of_cycles) do
19            node_belongs_to_a_cycle = false;
20            if (node belongs to one of set_of_cycles' cycles) then
21              push all cycle nodes to temp_maximal_concurrencies;
22              delete all cycle nodes from ons_nodes;
23              delete detect cycle from set_of_cycles;
24              node_belongs_to_cycle = true;
25              break;
26            end
27          end
28          if (node belongs to cycle = false) then
29            push node to temp_maximal_concurrencies;
30            delete nodes from ons_nodes;
31            break;
32          end
33        end
34      end
35    end
36  end
37  push temp_maximal_concurrencies to set_of_maximal_concurrencies ;
38 end
39 return set_of_maximal_concurrencies ;
40 End ;
```

Note that the algorithm always succeeds when applied to a well-formed SON thanks to the general properties of SONs established in [42]

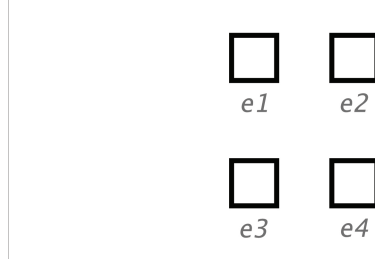


Figure 3.7: The result after detecting asynchronous communication.

3.7 Implementation and test results

We have implemented the policy-driven visualisation algorithm as a Java plug-in in the SONCraft toolkit (based on the WorkCraft platform) [1]. WorkCraft is a platform that provides a framework for the development and analysis of graph models [53]. We have created a menu (Retrieve Big Data) that has two features. As shown in Figure 3.8, the first sub menu is ‘Import SON data’, and the other is ‘Maximal ONs events’.

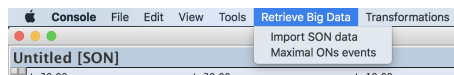


Figure 3.8: Retrieve Big Data menu.

Here is a brief explanation of the functionality of our plug-in. Firstly, our algorithm finds the starter conditions. These are conditions without pre-events. For every starter condition, we initialise two arrays. This will track which conditions will be added or removed for the next loop step. Next, we will find all cycles in the graph via the Tarjan Algorithm and store them in a list. Afterwards, we need to find starter events and store them in a list. Starter events are all events that do not have previous events. Next, we will loop through all starter conditions. If we catch

any starter events (for every event), we will add them to our temporary array if they have no connections. We will also add them to our temporary array if they have two connections, and the event does not follow an existing event in the temporary array. Then we will set the next condition as a starter and mark it if it is not in a cycle. However, if it is in a cycle, we will mark the starter condition and treat it separately. Next, we will loop through all starter conditions again. For every starter condition, we check to see if the condition is not being marked for removal (the condition has been skipped because the event following the condition is a part of Trajan's cycle). If this is the case, we obtain its post-event and all cycle events. Just after the two loops, we will decide if a set of maximal concurrency has completed or not. Finally, after we are sure that there are no more starter conditions, we will display the maximal concurrency events [24]. We now discuss the results obtained after the plugin

The output

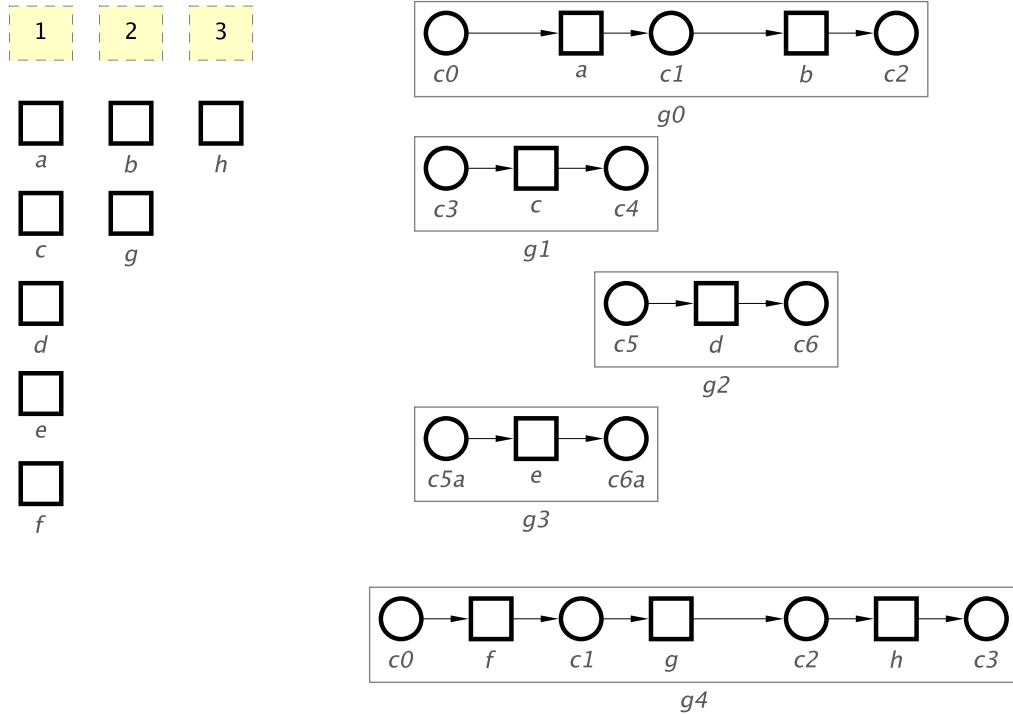


Figure 3.9: First test result.

was developed and implemented. The first test was a scenario of different ONs with

no communication between them. This shows that the plugin works correctly. For instance, in Figure 3.9, the first set comprises events $(a, c, d, e$ and $f)$. Afterwards, we obtain the next set consisting of two events $(b$ and $g)$. The last set has one event (h) .

Figure 3.10 provides another example of how our plugin treats complex scenario. It involves a model with three synchronous communications. As Figure 3.10 shows, the algorithm is working properly and the three maximal sets are $(a, d$ and $f)$, $(b, e$ and $g)$ and (c) .

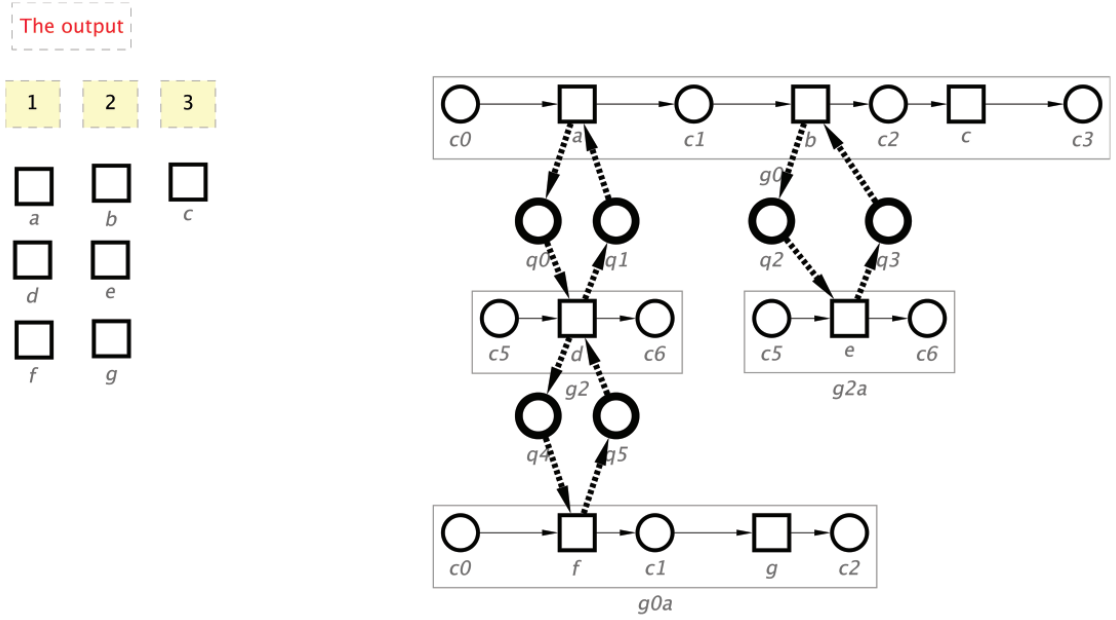


Figure 3.10: Second test result.

Figure 3.11 shows the last test case. It involves asynchronous communications, and the algorithm successfully pushes its events into one set, namely $(e2$ and $e4)$.

The output

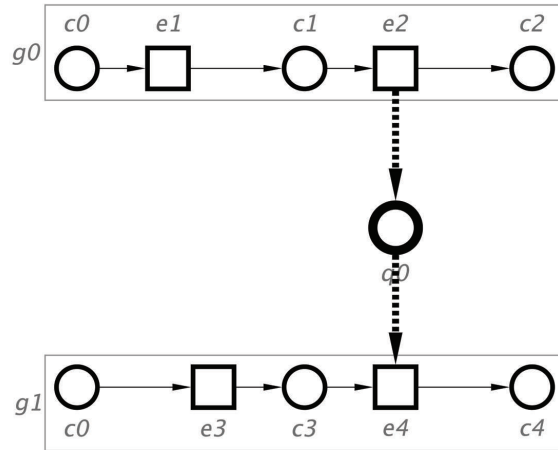
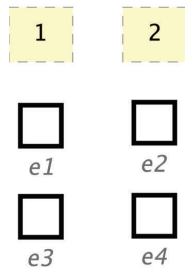


Figure 3.11: Third test result.

Finally, as displayed in the next figure 3.12, we provided a figure that shows a larger example of our plug-in.

3.8 Limitations of the proposed Visualisation Approach

Although the proposed approach has contributed significantly to the visualisation of the structure occurrence net (SON), the resulting visualisation approach has some limitations. In the case of dealing with uploading of big data, the user may find issues in the analysis of the data because the SONCraft is not able to deal with big data yet. An interesting extension would be to apply the current visualisation algorithms to other SON based models, such as behavioural abstractions. Finally, the SONCraft needs to be improved in terms of dealing with uploading data automatically and in real-time scenarios rather than just loading data from off-line files.

3.9 Concluding remarks

In this chapter, the problems of ONs overlapping and out-of-order placement have been looked into, and a novel solution to address them has been proposed. The design and development of the corresponding algorithm has been discussed. We provided test examples showing that our algorithm works properly.

In future work we will address the visualisation of behavioural abstraction through maximal concurrency policy. Another direction for future work is to implement visualisation based on local maximal concurrency [41]. It is a step execution policy that introduces an extension of the basic model of Petri nets, by adding the notion of located transitions and locally maximally concurrent executions of co-located transitions in order to represent the compartmentalisation of membrane systems [41]. The main idea depends on specifying the locality for transitions, each transition belonging to a fixed unique locality. The precise mechanism to achieve this is to introduce a partition of the set of all transitions using locality mapping [9].

Chapter 4

Domain Name System (DNS) tunnelling detection using Structured Occurrence Nets (SONs)

4.1 Summary

At present time, serious warnings regarding the increasing number of DNS tunnelling methods are on the rise. Attackers have used such techniques to steal data from millions of accounts. The existing literature has convincingly demonstrated the extent of the damage which DNS tunnelling can inflict on any given DNS server. Such threats can be alleviated, through SON based approach described in this chapter. The current chapter proposes the use of SON features with the purpose of detecting DNS tunnelling, in the event of an actual attack.

4.2 Introduction

In the past decades, internet usage and spread has grown dramatically, expanding to include everything and anything, from small, online businesses, to large company websites. As a consequence, what was initially ‘the Web’ became ‘Web 2.0’, considering the number of applications which has been developed, based on the internet. It is the Domain Name System or DNS, which allows applications to use names instead of numbers (IP addresses), which are considerably more difficult to deal with. There are several works [49, 67, 18, 7] warning of the increasing number of DNS tunnelling methods, and attackers have used these techniques to compromise millions of accounts [31]. The existing literature has demonstrated the damage which DNS tunnelling can make on any deployed DNS server. DNS tunnelling allows hackers to transfer data in a way which violates established system security policies [7]. The danger which such an attack brings is that it can occur without triggering any alarms. It is actually shown as a legitimate activity, because it uses a DNS protocol to transmit information in the original way, instead of focusing on DNS vulnerabilities or abusing or exploiting the system.

The reason behind using SONS to record DNS traffic are the different features that SONS provide. For example, a SON combines multiple related ONS by using various formal relationships, in particular, for communication. For instance, SON events can be used to model various types of packets, such as sending a query, receiving a query, sending a response and receiving a response, as well as asynchronous communication. By means of these features, a SON is able to depict and analyse a DNS case. The approach which we propose is to use SON features to detect DNS tunnelling in the event of a real attack. This chapter is organised as follows: Section 4.2 introduces the problem. Section 4.3 describes the background concerning DNS. Within Section 4.4 the DNS tunnelling tools discussed. Section 4.5 focuses on preprocessing datasets stage. Section 4.6 describes a novel solution used to detect DNS tunnelling. In

Section 4.7 discussed the results and evaluation of the proposed algorithm. Section 4.8 provides concluding remarks.

4.3 Authoritative name server

An authoritative name server only provides replies to DNS queries from data which has been configured through an original source, for instance either by the domain administrator or other dynamic DNS methods, as opposed to answers received via queries made to other name servers which only maintain a small data cache. There are two types of authoritative name servers: master or slave servers. A master server stores all original records. By contrast, a slave server uses an automated updating mechanism, as part of its DNS protocol, and communicates with the master server, with the goal of maintaining identical copies of all records. A range of authoritative name servers are assigned to each DNS zone. Such a range is stored within a parent domain zone which is found within the name server (NS). In addition, an authoritative server only provides ‘authoritative answers’, namely definitive answers, marked as a protocol flag [51]. The protocol flag usually features in the the output of DNS administration query tools, with the purpose of indicating that the responding name server is indeed an authority for the respective domain [51].

We will use terminology presented in Table 4.1.

Table 4.1: Terminology used in the discussion of DNS tunnelling.

No.	Term	Description
1	SONs	Structured Occurrence Nets
2	ONs	Occurrence Nets
3	DNS	Domain Name System
4	HTTP	HyperText Transfer Protocol
5	FTP	File Transfer Protocol
6	SSH	Secure Shell
7	IP	Internet Protocol
8	iodine	Software that lets you tunnel IPv4 data through a DNS server















Steps of DNS Tunnel		
 COMPROMISED PC	Wants to send data "secret" to the malicious server Firewall blocks direct communication !!!	 FIREWALL
 COMPROMISED PC	Base64 encodes the message "secret" secret → c2VjcmV0	
 COMPROMISED PC	Q-1: What is the IP of c2VjcmV0.talal.com? A-1: (Wait a second)	 INTERNAL DNS SERVER
 INTERNAL DNS SERVER	Q-2: What is the IP of .com? A-2: IP is A.B.C.D	 DNS ROOT SERVER
 INTERNAL DNS SERVER	Q-3: What is IP of DNS server that is responsible for the DNS Zone "talal.com" ? A-3: IP is E.F.G.H	 .COM TOP LEVEL DOMAIN SERVER (DNS)
 INTERNAL DNS SERVER	Q-4: What is the IP of c2VjcmV0.talal.com? A-4: Returns a dummy answer: 127.0.0.1	 talal.com DNS SERVER
 ALEX.COM'S DNS SERVER	Analyzes the query: → c2VjcmV0.talal.com → c2VjcmV0 → Base64Decode c2VjcmV0 → secret	Data is extracted from within the DNS query.
 INTERNAL DNS SERVER	A-1: 127.0.0.1	 COMPROMISED PC

Figure 4.1: DNS tunnelling steps.

4.4 DNS tunnelling tools

There are several tools that can be used by hackers to tunnelling the DNS protocol. Most of these tools run under the Linux operation system, as describe below:

1. **DeNise** developed in Python is a proof of concept for tunnelling the TCP over DNS.
2. **dns2tcp** is a network tool designed to rely on TCP connection via DNS traffic. It contains two parts, first a server-side tool and second a client-side. In the server, there is a list of resources; each resource is local or remote services that listen to for TCP connections. However, the client listens to the TCP port and relays each income connection via DNS [11].
3. **DNScapy** is a DNS tunnelling tool written in Python. It also has server-side and client-side. However, server one can control multiple clients. It is created an SSH tunnel via DNS packets [13].
4. **DNScat** is designed to create an encrypted command and control (C&C) channel over the DNS protocol. It is written in C [14].
5. **Heyoka** is an exfiltration tool that uses to proof of concept that spoofed DNS requests to create a cover channel. Although this tool is fast, it is not under development. It was tested on Windows [21].
6. **Iodine** is a DNS tunnelling tool. It is working under Linux. It allows users to tunnel IPv4 data through a DNS server. This tool can be used to internet access is fire-walled, but DNS queries allowed. It also provides high performance. As well as portability which means it can be run on Win 32 side by side with Linux. The iodine security is high by MD5 hash and can be filtered out any particular packets that not come from the IP use when logging in. The other feature with iodine is that there is less step to create successful tunnelling [16].

We have focused on Iodine which is, as far as we know, a popular one among the attackers.

4.5 Preprocessing datasets

Data collection constitutes the initial stage of our work. The first step consists of discovering how DNS packets work, how they behave and what their structure is. This step actually consists of two stages which have been conducted in parallel: first, the extensive studying of literature pertaining to the subject and second, capturing real data from a local DNS server. Initially, we captured normal DNS packets. Afterwards, we modelled the normal packet in our model SON. Table 4.2 illustrates the DNS packet and Figure 4.2 shows the result. This type of data can be modelled in SONs;

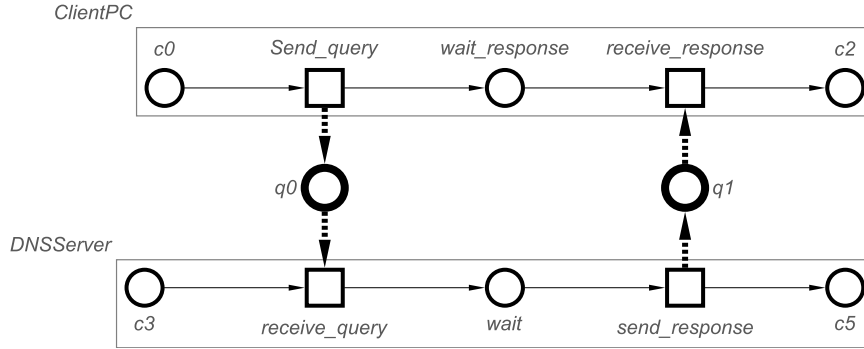


Figure 4.2: Normal DNS packet SON model

however, the ‘Info’ field has various information which must actually be split into different fields. The reason behind this split is to distinguish whether the data packet is either query or response. Additionally, this split serves to distinguish the domain names, but also to determine the packets id (0x81fe), since it is this ID which is our indicator, namely the one which links each packet (query) with its response. For instance, in table 4.2 row 26 is the query and it is linked with row 29, which is the response for the ID packet in question (0x81fe). We developed a python script [3] to deal with this particular scenario. The main idea of this script is to split the ‘Info’

Table 4.2: DNS Packets

No.	Source	Destination	Info
26	172.20.10.2	172.20.10.1	query 0x81fe A google.com
29	172.20.10.1	172.20.10.2	response 0x81fe A google.com

Table 4.4: DNS Tunnelling Packet

No.	Source	Destination	Operation	PacketID	Domain
1	172.20.10.2	146.185.138.197	query	0xc7cb	paaac5ay.tunnel.carn.us.to
2	146.185.138.197	172.20.10.2	response	0xc7cb	paaac5ay.tunnel.carn.us.to

field into 3 subfields, namely operation (which is either query or response), ‘PacketID’ and finally, ‘Domain’, as shown in the table 4.3. Afterwards, we modelled the last result of a normal packet, to ensure how it will be represented within the SON, as shown in Figure 4.2.

Table 4.3: Normal DNS Packet

No.	Source	Destination	Operation	PacketID	Domain
1	172.20.10.2	172.20.10.1	query	0x81fe	google.com
2	172.20.10.1	172.20.10.2	response	0x81fe	google.com

Moreover, during the experiments, we collected attack packets, the result of DNS tunnelling attempts. This was, in fact, the second challenge which we discovered and treated. Table 4.4 represents one such attack packet. As noticeable, if we attempt to model this packet, it will look the same as the normal packet, described above. Due to this, the point of the research is to discover an idea or algorithm which allows the clear distinguishing between normal packets and attack packets. The idea is one must examine each particular domain and its sub-domains, with the purpose of discovering whether or not a specific packet is an attack packet or not. As such, the python script was updated to split the fields with which we are dealing with, in this case. The main idea for the updated script version is that it loops on each domain field and links it by a unique ID. It is called GroupID. As a result, afterwards, we can identify each chunk of the domain and its sub-domains by that GroupID, as Table 4.5 illustrates. Table 4.5 illustrates the link between the ‘Google’ domain and its subdomains. For

Table 4.5: Mixed Packets (normal and abnormal packets)

No.	Source	Destination	Operation	GroupID	Domain
1	172.20.10.2	172.20.10.1	query	001	google.com
2	172.20.10.1	172.20.10.2	response	001	google.com 216.58.206.206
3	172.20.10.2	172.20.10.1	query	001	maps.google.com
4	172.20.10.1	172.20.10.2	response	001	maps.google.com
5	172.20.10.2	146.185.138.197	query	002	paaac5ay.tunnel.carn.us.to
6	146.185.138.197	172.20.10.2	response	002	paaac5ay.tunnel.carn.us.to
7	172.20.10.2	146.185.138.197	query	002	paaydani.tunnel.carn.us.to
8	146.185.138.197	172.20.10.2	response	002	paaydani.tunnel.carn.us.to

instance, we assume ‘google.com’ and ‘maps.google.com’ have the same group id, namely ‘001’. However, subdomains (paaac5ay) and (paaydani) have been grouped to ‘.us.to’ via group id, which is actually ‘002’. So, the idea is to initially identify the domain, i.e. ‘google’ and loop to find any particular sub-domain of ‘google’ and link it with a unique group id.

4.5.1 Mixing data

In order to simulate a real scenario of DNS server behaviour, the collected data consisted of two main parts. The first part included the data collected during the attack experiment. The second part includes normal data, collected from the university. For this data to make sense, and in order to simulate a real scenario example, we created python script [2] mixing these two kinds of data. This data mix will be evaluated in different stages in the evaluation section (4.7).

4.6 Proposed solution

4.6.1 Detecting DNS tunnelling using SONs

The main idea of our algorithm [4] to detect DNS tunnelling is to model DNS data i.e., each particular transaction and the way it will communicate with the local DNS server. We assume that each packet is a unique ON. And the local DNS server is one ON. Each packet (ON) communicates with the local DNS server by channel place (thick circle) via events (*Send – query* and *receive – query*), as shown in figure 4.2. The client (ONc) sends a query to local DNS server (ONserver), and then DNS server responds to the client as shown in Figure 4.2. This is the normal packet. When we were to model the attack packet, we found it is the same as the normal one in terms of communication and behaviour. The reason is that we examined each packet as a separate one. However, to detect an attack we need to examine each domain and its sub-domains in order to examine either we see normal or abnormal packets. For instance, if google.com, map.google.com and developers.google.com are involved, then we need to deal with all google domains and its sub-domains, and model them as one of chunk of domain packets. So, we group each domain and its sub-domains with unique ID as in preprocessing section above. The result is seen in Figure 4.3, where we can distinguish between normal and abnormal packets (we displayed the attack ON very small to fit the width of the text; however, the ON transaction is actually bigger).

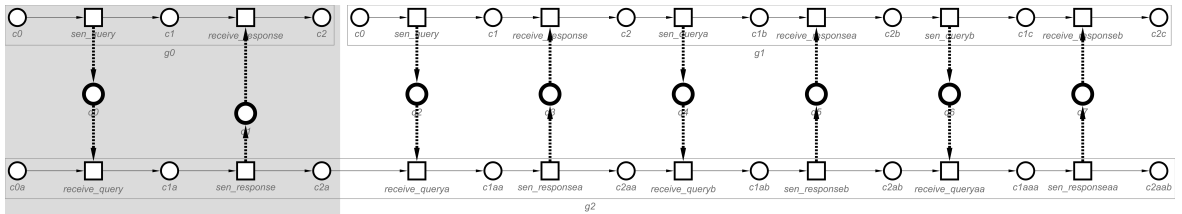


Figure 4.3: Normal and Abnormal ONs

4.6.2 Detection DNS tunnelling algorithm

The main idea behind the algorithm is first to detect any ON input, whether it is a normal input or an ON representing a DNS attack. Before going more in depth, it should be mentioned that ONs refer to DNS packets. Furthermore, we assume the local DNS local server to be another ON. It does not matter whether inputs are normal or representing a DNS attack. We examine each ON input and count its events which communicate with the DNS Server ON. If the number of those ON events is less than threshold value, we flag it as a normal ON. Otherwise, we flag it as an abnormal ON. Then, we check whether or not we had abnormal ONs; if so, then we know a DNS attack has occurred. Finally, we should mention that the aforementioned threshold was identified via logistic regression, a widely used statistics model. The idea behind this binomial model resides in properly estimating the parameters of the logistics model. From a mathematical standpoint, the model relies on variables with two possible values, for instance ‘win’ or ‘lose’ or ‘pass’ or ‘fail’. As such, we applied logistic recognition to available data in order to estimate the threshold, and estimate how many events within a particular GroupID could be suspicious.

4.6.2.1 The calculated threshold

It is at the core of the logistic regression model. From a mathematical standpoint, logistic regression performs multiple linear regression functions of the required feature, such as the number of events. These functions are based on the dependent, dichotomous variable, i.e. ‘attack’ or ‘no attack’. The aforementioned functions produce a threshold by using the *logit* equation. Finally, this threshold can be used in our algorithm. The mathematical aim of linear regression functions is to build an equation with the following coefficients: (a) and (c), as follows: $y = ax + c$, where (y) represents the probability of an attack and (x) is the number of events. Afterwards, the values for both the number of events and the dependent variable, namely ‘attack’

or ‘not attack’, will be inserted; this will allow us to perform regression analysis, meaning we will be able to calculate the values of (a) and (c), which in turn will increase the likelihood of (y) conforming to ground truth values. Once these values are known, they can be used to calculate the threshold.

ALGORITHM 2: Detection DNS Tunnelling algorithm using SON

Input: set_of_ons : Set of All Ons

dns_server_on : DNS Server ON

attack_max_events : DNS Server threshold to interpret an attack

Output: set_of_normal_ons = Set of normal ONs, not presenting a potential attack.

set_of_abnormal_ons = Set of abnormal ONs, presenting a potential attack.

```
1 Interpretation:
   if size of (set_of_abnormal_ons) > 0 then
2 |   DNS Attack;
3 else
4 |   No DNS Attack;
5 end
6 Start ;
7 array set_of_normal_ons = [] ;
8 array set_of_abnormal_ons = [] ;
9 integer set_of_normal_ons_index = 0 ;
10 integer set_of_abnormal_ons_index = 0 ;
11 integer connected_events = 0 ;
12 for (count_event(current_on)) < attack_max_events)) do
13 |   connected_events_count = 0;
14 |   if (count_event(current_on)) < attack_max_events)) then
15 |       set_of_normal_ons[set_of_normal_ons_index] = current_on;
16 |       set_of_normal_ons_index = set_of_normal_ons_index + 1;
17 |       continue;
18 |   else
19 |       for (Event current_event in current_on's events) do
20 |           if (current_event is connected to dns_server_on) then
21 |               | connected_events_count = connected_events_count + 1;
22 |           end
23 |       end
24 |       if (connected_events_count >= attack_max_events) then
25 |           set_of_abnormal_ons[set_of_abnormal_ons_index] =
                current_on; set_of_abnormal_ons_index =
                set_of_abnormal_ons_index + 1;
26 |       else
27 |           set_of_normal_ons[set_of_normal_ons_index] =
                current_on; set_of_normal_ons_index =
                set_of_normal_ons_index + 1;
28 |       end
29 |   end
30 end
```

```

1 return structure : {set_of_normal_ons , set_of_abnormal_ons} ;
2 count_event(ON on) function ;;
   Input: on : ON
   Output: events_count : number of events of ON.;
3 Start ;
4 integer vents_count = 0 ;
5 for (Event current_event in current_on's events) do
6 | events_count = events_count + 1 ; ;
7 end
8 return events_count ;
9 End ;

```

4.7 Result testing and evaluation

We considered the following proportions of attacking packets (as percentage of the total packets):

0%, 1%, 5%, 10%, 20%, 40%, 60%, 80%, 90%, 95%, 99% and 100%.

In addition, we used the sensitivity and specificity methods [35], developed to evaluate a system of computer-assisted diagnosis. Hence we used a tried and tested method, with the only change being made to the terminology, as follows:

First, True positive (TP) result: attack transactions were correctly identified as attacks. Then, False positive (FP) result: normal transactions were incorrectly identified attacks. And, True negative result (TN) : normal transactions were identified as normal transactions. Finally, False negative (FN) result: attack transactions were incorrectly identified as normal transactions. In terms of the data mix (of normal and attack packets), we created a Python script [2]. The idea behind the script was to mix the two data sets, namely: the normal and attack packets. Each time we ran the script, we asked the script to mix the data, relying on the percentage from each file, such as 10% normal and 90% attack; as a result, we obtained one file with mixed data. The main data sets for each packet type, i.e., normal or attack packets, were collected during a five-minute time span. The data for both sets consisted of

roughly 700 DNS packets. After running the algorithm, we checked the sensitivity and specificity through an evaluation method, obtaining the following results. As evident in (1) in Table 4.6, in True Positives and False Negatives, we assumed that there was no result because this data set had no attack packets. However, in (2) in Table 4.6, when the data set had 1% attack and 99% normal packets, we got a 0% True Positive, which means there were no attacked packets detected as attack packets, although we had 1% attack data. This is because the number of packets in this data set was less than the threshold. Regarding False Negatives, we also got 0% because there were no attack packets detected as normal packets. Likewise, in (3) in Table 4.6, the same situation applied. However, in (4) in Table 4.6, the algorithm detected 7.1% from a total of 10% of the attack data; however, it failed to detect 2.9% of the attack packets. After we checked the data sets manually, we found that most of these packets were only request queries to the DNS server, to which the DNS server did not respond, so our algorithm skipped them. However, from this 2.9%, the algorithm did not attacked them, although they have query and response packets. In future research, we will further investigate this issue in order to discover why this happened. Another interesting point is that, in (4) in Table 4.6, there is a False Positive, which means the algorithm detected 9.4% of the normal packets as attack packets; after investigating this, we found that Google had sent many requests to the DNS server as unusual requests, and these request events were above our threshold. When we rerun the experiment, we did not face this issue. Moreover, as evident in (7) in Table 4.6, no normal packets were detected as attack packets (False Positives). Finally, in (15) in Table 4.6, the False Positives and True Negatives were ‘N/A’; this was because, in these data sets, there were no normal packets at all.

Table 4.6: Table of results

No.	% normal and abnormal packets	TP	FP	TN	FN
1	0% attack, 100 normal	N/A	9.7%	90.92%	N/A
2	1% attack, 99 normal	0%	9.77%	90.92%	0%
3	5% attack, 95 normal	0%	8.9%	91.1%	0%
4	10% attack, 90 normal	7.1%	9.4%	90.6%	2.9%
5	20% attack, 80 normal	82.9%	5.3%	94.7%	17.1%
6	30% attack, 70 normal	86.7%	5.3%	94.7%	13.3%
7	40% attack, 60 normal	88.5%	0%	100%	11.5%
8	50% attack, 50 normal	90.9%	0%	100%	9.1%
9	60% attack, 40 normal	91.5%	0%	100%	8.5%
10	70% attack, 30 normal	91.1%	0%	100%	8.9%
11	80% attack, 20 normal	91.5%	0%	100%	8.5%
12	90% attack, 10 normal	91.8%	0%	100%	8.2%
13	95% attack, 5 normal	92%	0%	100%	8%
14	99% attack, 1 normal	92.3%	0%	100%	7.7%
15	100% attack, 0 normal	92.5%	N/A	N/A	7.5%

4.8 Concluding remarks

This chapter described a novel solution to DNS tunnelling detection based on SONS. An detection algorithm has been designed and implemented. Also, data preprocessing and a set of experiments have been discussed. Further work will focus on improving the current algorithm, aiming at allowing it to work automatically in terms of reading packets in real-time scenarios. In addition, we will model and develop algorithms dealing with more than one DNS server at a time, including all user packets. In addition, we will use other DNS tunnelling tools, instead of ‘iodine’, in order to compare various attack behaviours and check how the algorithm deals with these types of attacks.

Chapter 5

Visualisation and Abstract Conditions (Extensions)

5.1 Summary

In this chapter, we present a new class of SONS based on multi (coloured) tokens. This extension was motivated by the result we obtained in Chapter 4. Although the results were exemplary, the limited screen size and large number of transactions (DNS packets) meant that we were able to visualise only a small part of data. Hence limitations existed in terms of visualisation. To address this challenge, we will provide a new condensed/abstract representation of a large number of similar concurrent processes so that the same underlying net structure can be used to model their behaviour. This arrangement makes the analysis of the overall behaviour more efficient and its visualisation more effective (and, in many cases, simply feasible).

5.2 Introduction

After describing the methods, techniques and algorithms used in this thesis, we will now introduce a new class of Structured Occurrence Nets (SONs). We see this exten-

sion as the beginning of new techniques and implementations that will be using big data and integrate with SONCraft in future.

Today's world deals with vast amounts of data that we store in offline storage, such as hard drives, local servers or the web. All technology today must handle that data, whether it relates to e-commerce, health or even crime. Managing data on such a large scale is a challenge, both academically and industrially. As mentioned in the previous chapters (Chapters 3 and 4) the SON framework provides new and interesting results related to the analysis of cybercrime investigations. However, because of the inherent features of SONs (only one token can be present in a condition at any time), it lacks in ability to present large amounts of data. The idea behind discussed extensions is to enhance SONs to be able to visualise the large data concisely. Our aim in extending conditions to accept more than one token is that, while working on Chapter 4, we successfully obtained an interesting result in the detection of DNS tunnelling. In order to visualise groups of DNS packets — because these packets are identical within the model — we have to repeat each one as a unique model, as shown in Figure 5.1.

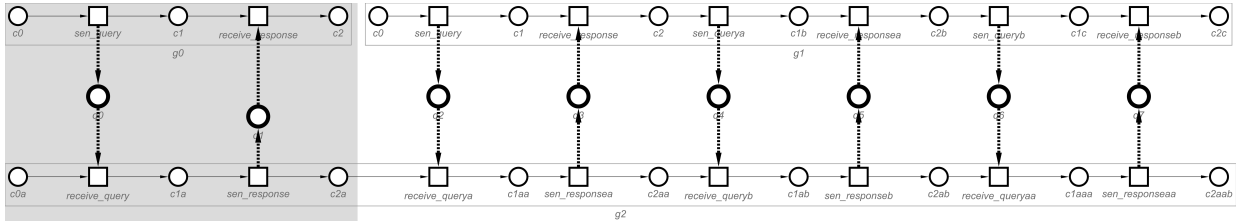


Figure 5.1: Normal and Abnormal ONs.

As shown, there are four normal packets that behave in the same way. Our idea is that, if all the above packets behave in the same way, then it is suitable to make them a single mode in one model and to refer to each behaviour by a token rather than by a model (net structure). The multiple tokens will enable us to use multiple model behaviours within a single model when all the models are identical. The extension will focus not only on providing new standard tokens but will provide a way to use the well-known Coloured Petri Net (CPN) technique. CPN is a graphics-oriented

language used to design and specify simulation and verification of systems. It is an extension of the concept of the mathematical model of Petri Nets (PNs). CPNs allow distinctions between tokens. In other words, CPNs allow tokens to have attached data values. These attached values are known as token colour which can come from an arbitrarily complex data type.

This chapter is organised as follows: Section 5.3 provides an overview of the big data. Section 5.4 provides an overview of general Petri Nets (PNs). After that, Section 5.5 provides an overview of Coloured Petri Nets (CPNs). An extension of the notion of CSON is given in Section 5.6. Section 5.7 will discuss and provide different examples of coloured tokens in SONS. Finally, the conclusion and future work is presented in Section 5.8.

5.3 Big Data

Massive data is used on the internet in today's world. The amount of data will increase by 2025 [17] so that it will cross the total brain size of people. This growth will come from advances in digital sensors, communications and storage, which will create a massive amount of data [10]. As a result of this growth, we will focus on big data in this section, including the definition of big data and big data management. Then we will focus on how big data can be handled within SONS.

Big data [48] is any particular data set the size of which outweighs the normal ability of data set tools. Additionally, in [10], big data is defined as too big to be controlled, operated or analysed by normal database concepts such as Structured Query Language (SQL). To be considered big data, any particular data set must have the three Vs: volume, variety and velocity [17]. The main benefit of supporting big data is to allow us to analyse big data sets, such as those on health, biochemistry, genetics and criminal activity, in order to carry out effective analysis and arrive at

new knowledge. The other benefit is variety; this kind of data does not have a fixed structure, which gives us a chance to present it in our approach in most helpful way for our analysis. Big data can be organised, for instance, in relational databases, or in semi-structured ways such as web logs, social media feeds, emails or sensors. Big data architecture needs high-speed data from different sources including web, NoSQL and DMBS, and it must be dealt with using diverse access protocols. In some applications, the generation of data is required to generate important data, and then it may be interesting to obtain further analysis and capture these metadata to store with similar data.

An appropriate visualisation of big data sets will lead to clearer images for criminal investigators and potentially allow them to link crime events and places in a representation of data that helps detect crimes. The results of such research could also be an essential step toward crime prevention. Another benefit of using big data is the extraction of valuable knowledge from big and complex data sets, and the identification of useful information retrieved from various data sources. In addition, data can be explained as an automated extraction of predictive information from large sets of data.

5.4 Petri Nets (PNs)

(General) Petri Nets (PNs) were introduced by Carl Adam Petri in 1962 in order to model concurrent behaviour within a system. A PN is triple $PN = (P, T, \varphi)$ where P is a finite set of places, and T is a finite set of transitions. These two sets are disjoint. The φ is a flow function from $(P \times T) \cup (T \times P)$ to \mathbb{N} . A *marking* is mapping $M : P \rightarrow \mathbb{N}$. $M(p)$ gives the number of tokens in each place p of PN . PNs are directed graphs that have two types of nodes: places are represented by circles, transitions are represented by bars or boxes, and tokens are represented by small

black circles. Each arc refers to $\varphi(p, t)$ or $\varphi(t, p)$, where p and t are the endpoints of the arc. Figure 5.2 show a PN example.

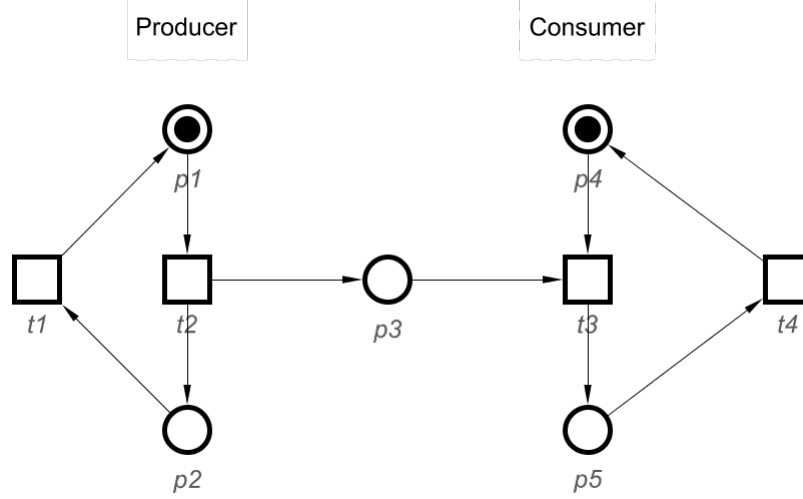


Figure 5.2: Petri Net example.

A transition $t \in T$ is enabled at marking M if for every $p \in P$, $\varphi(p, t) \leq M(p)$ [70]. Moreover, if the transition t is enabled, then it may be fired by removing $\varphi(p, t)$ tokens from each place p and placing $\varphi(t, p')$ tokens in every place p' . For example, the transition $t2$ in Figure 5.2 is enabled at marking $M = (1, 0, 0, 1, 0)$, as the only input place of $t2$ satisfies $\varphi(p1, t2) \leq M(p1)$. After firing the transition $t2$, the PN will reach a new marking $M' = (0, 1, 0, 1, 0)$.

5.5 Coloured Petri Nets (CPNs)

Coloured Petri Nets (CPNs) [28] is a graphics-oriented language for the design, simulation and verification of concurrent systems. It is an extension of PNs, which is well suited to systems that support communication and synchronisation between the components of systems. CPNs can be used for communication protocols, embedded systems, workflow analysis, and distributed systems.

In our particular case, CPNs have four main components. Places (represented by circles), transitions (represented by rectangles or boxes), and arcs (represented

by arrows), describe the operation of changes to the state of CPN when transitions happen. The fourth component of CPNs are tokens, which are a set of markers to be placed inside the places. A main difference with PNs is that each of these tokens has a data value that comes from a given data *type*.

5.6 Extensions

What we would like to do is extend the notion of CSON to cover the situation when more than one concurrent process is executed using the same *static net structure* of a CSON. This will be done in two different ways. However, before doing so we need to clarify what could be behavioral properties which any such extension is required to satisfy.

Basically, what we want to achieve is a condensed/abstract representation of perhaps a large number of similar concurrent processes (so that the same underlying net structure can be used to model their behaviour), making the analysis of the overall behaviour more efficient and its visualisation more effective (and in many cases simply feasible).

We first introduce an extension of CSONs based on ‘coloured tokens’ and ‘coloured events’ which is an instance of a well-established Petri net technique of coloured (or high-level) nets [43].

Definition 5.6.1 (CCSON) *A coloured communication structured occurrence net (CCSON) is a tuple*

$$\text{CCSON} = (ON_1, \dots, ON_k, Q, W, \text{col})$$

such that

$$\text{CSON} = (ON_1, \dots, ON_k, Q, W)$$

is a CSON as in Definition 2.3.3 and $\text{col} \neq \emptyset$ is a finite set of colours.

- The coloured tokens and coloured events of CCSON are respectively:

$$\begin{aligned} C_{col} &= (\mathbf{C} \cup Q) \times col \\ E_{col} &= \mathbf{E} \times col \end{aligned}$$

- A marking of CCSON is a set of coloured tokens, and the initial marking of CCSON is $M_0^{\text{CCSON}} = M_0^{\text{CSO}} \times col$.
- For each $(e, c) \in \mathbf{E} \times col$,

$$\begin{aligned} \bullet(e, c) &= \bullet e \times \{c\} \\ (e, c)\bullet &= e\bullet \times \{c\}. \end{aligned}$$

Moreover, for every $U \subseteq \mathbf{E} \times col$,

$$\begin{aligned} \bullet U &= \bigcup_{(e, c) \in U} \bullet(e, c) \\ U\bullet &= \bigcup_{(e, c) \in U} (e, c)\bullet. \end{aligned}$$

- For each colour $c \in col$, the set $\{e \mid (e, c) \in U\}$ is a step of CSO (see, for example an enabled step $U = \{(e0, B), (e1, B)\}$ in Figure 5.14)
- A step U is CCSON-enabled at a marking M if $(\bullet U \setminus U\bullet) \subseteq M$.
- If U is CCSON-enabled at a marking M , then U can be fired and produce a new marking M' given by

$$M' = (M \cup U\bullet) \setminus \bullet U.$$

This is denoted by $M[U]_{\text{CCSON}} M'$.

- A step sequence execution of CCSON is a sequence

$$\gamma = M_0 U_1 \dots M_{n-1} U_n M_n$$

such that $M_0 = M_0^{\text{CCSON}}$ and $M_{i-1}[U_i]_{\text{CCSON}} M_i$, for $i = 1, \dots, n$.

Intuitively, every colour $\mu \in \text{col}$ corresponds to a separate process running on the net structure of CCSON.

To show that the behaviours of CCSON are strongly related to the behaviours of CSON, we introduce a projection operation from colored tokens and events to the original conditions and events. For every set X of coloured tokens and/or events, and for every colour $\mu \in \text{col}$, we denote:

$$X \downarrow \mu = \{x \mid (x, \mu) \in X\}.$$

Theorem 5.6.1 *Assume the notation as in Definition 5.6.1.*

1. Let $\gamma = M_0 U_1 \dots M_{n-1} U_n M_n$ be a step sequence execution of CCSON. Then, for every $\mu \in \text{col}$,

$$\gamma_\mu = (M_0 \downarrow \mu) (U_1 \downarrow \mu) \dots (M_{n-1} \downarrow \mu) (U_n \downarrow \mu) M_n \downarrow \mu$$

is a step sequence execution of CSON. Moreover, for $i = 0, 1, \dots, n$ and $j = 1, \dots, n$, we have:

$$M_i = \bigcup_{\mu \in \text{col}} (M_i \downarrow \mu) \times \{\mu\} \text{ and } U_j = \bigcup_{\mu \in \text{col}} (U_j \downarrow \mu) \times \{\mu\}.$$

2. Let $\text{col} = \{\mu_1, \dots, \mu_l\}$ and, for $i = 1, \dots, l$, let

$$\gamma_i = M_0^i U_1^i M_1^i \dots U_n^i M_n^i$$

be a step sequence execution of CSON.¹ Then

$$\gamma = (\bigcup_{i=1}^l M_0^i \times \{\mu_i\})(\bigcup_{i=1}^l U_1^i \times \{\mu_i\}) \dots (\bigcup_{i=1}^l U_n^i \times \{\mu_i\})(\bigcup_{i=1}^l M_n^i \times \{\mu_i\})$$

is a step execution of CCSON.

Proof: The result follows directly from definitions. □

Part (1) of the above theorem means that every step sequence execution of CCSON can be seen as a combination of a number of step sequence executions of the underlying CSON. Part (2) states the opposite, namely that for a set of step executions of the underlying CSON, for each available colour, one can combine them into a step sequence execution of CCSON.

Using colours to distinguish similar processes running over the same net structure is an abstraction technique which improves the visualisation of the overall system behaviour. In particular, coloured tokens can be placed in the circles.

The next definition abstracts sets of coloured tokens to multisets of ordinary tokens (conditions), and sets of coloured events to multisets of events. In this way, one ‘forgets’ about the colours and only counts the number of processes which have their local states in a given condition at any time. This is highly advantageous both for visualisation and verification.

A multiset over a set X is a function $\tau : X \rightarrow \mathbb{N}$. If τ and τ' are multisets over X , then we define $\tau + \tau'$, $\tau - \tau'$ and $k \cdot \tau$ as multisets over X such that, for every $x \in X$:

$$\begin{aligned} (\tau + \tau')(x) &= \tau(x) + \tau'(x) \\ (\tau - \tau')(x) &= \max\{\tau(x) - \tau'(x), 0\} \\ (k \cdot \tau)(x) &= k \cdot \tau(x). \end{aligned}$$

¹We can assume that the step sequence executions γ_i have the same length. This can always be ensured by padding the shorter executions with empty steps.

Definition 5.6.2 (MCSON) A multiset *communication structured occurrence net* (MCSON) is a tuple

$$\text{MCSON} = (ON_1, \dots, ON_k, Q, W, m)$$

such that $m \geq 1$ and

$$\text{CSON} = (ON_1, \dots, ON_k, Q, W)$$

is a CSON as in Definition 2.3.3.

- A marking of MCSON is a multiset over $\mathbf{C} \cup Q$, and the initial marking of MCSON is $M_0^{\text{MCSON}} = k \cdot M_0^{\text{CSON}}$.
- A step of MCSON is a multiset over \mathbf{E} .
- A step U is MCSON-enabled at a marking M if, for every $c \in \mathbf{C} \cup Q$:

$$M(c) \geq \sum_{e \in c^\bullet} U(e) - \sum_{e \in {}^\bullet c} U(e).$$

- If U is MCSON-enabled at a marking M , then U can be fired and produce a new marking M' given, for every $c \in \mathbf{C}$, by:

$$M'(c) = M(c) - \sum_{e \in c^\bullet} U(e) + \sum_{e \in {}^\bullet c} U(e).$$

This is denoted by $M[U]_{\text{MCSON}} M'$.

- A step sequence execution of MCSON is a sequence

$$\gamma = M_0 U_1 \dots M_{n-1} U_n M_n$$

such that $M_0 = M_0^{\text{MCSON}}$ and $M_{i-1}[U_i]_{\text{MCSON}} M_i$, for $i = 1, \dots, n$.

We will now show that the behaviours of multiset based CSONs and the behaviours of coloured CSONs are closely related.

Theorem 5.6.2 *Assume the notations as in Definitions 5.6.1 and 5.6.2 and assume that $m = |\text{col}|$.*

1. *Let $\gamma = M_0 U_1 \dots M_{n-1} U_n M_n$ be a step sequence execution of CCSON. Then:*

$$\gamma' = M'_0 U'_1 \dots M'_{n-1} U'_n M'_n$$

is a step sequence execution of MCSON, where, for all $i = 0, 1, \dots, n$, $j = 1, \dots, n$, $p \in \mathbf{C} \cup Q$, and $e \in \mathbf{E}$:

$$\begin{aligned} M'_i(p) &= |\{\mu \mid (p, \mu) \in M_i\}| \\ U'_j(e) &= |\{\mu \mid (e, \mu) \in U_j\}|. \end{aligned} \tag{5.1}$$

2. *Let $\gamma' = M'_0 U'_1 \dots M'_{n-1} U'_n M'_n$ be a step sequence execution of MCSON. Then there are step sequence executions*

$$\hat{\gamma}^j = \widehat{M}_0^j \widehat{U}_1^j \dots \widehat{M}_{n-1}^j \widehat{U}_n^j \widehat{M}_n^j \quad (1 \leq j \leq m)$$

*of CSON such that, for $i = 0, 1, \dots, n$ and $k = 1, \dots, n$:*²

$$M'_i = \sum_{j=1}^m \widehat{M}_i^j \text{ and } U'_k = \sum_{j=1}^m \widehat{U}_k^j.$$

3. *Let $\gamma' = M'_0 U'_1 \dots M'_{n-1} U'_n M'_n$ be a step sequence of MCSON. Then there is a step sequence execution of CCSON*

$$\gamma = M_0 U_1 \dots M_{n-1} U_n M_n$$

such that the formula (5.1) is satisfied.

²In the formula below, we treat sets as special kind of multisets.

Proof: Part (1) follows from a straightforward induction on the length of the sequence. Part (2) can be shown using Theorem 18 in [66] which essentially proves the required property (called there serialisability) for acyclic marked graphs, where each place has at most one input transition and at most one output transition. Both these properties hold for MCSN, and it is possible to convert (for the purpose of establishing the result) MCSN into an equivalent acyclic marked graph (by gluing together clusters of synchronised transitions). Part (3) follows from Part (2). \square

Parts (2,3) of the above result mean that the behaviours of MCSN can be factorised onto behaviours of CSN.

5.7 Examples of coloured tokens in SONs

This section provides some examples of the proposed extension to include coloured tokens. First, consider an occurrence net $ON1$ shown in Figure 5.3.

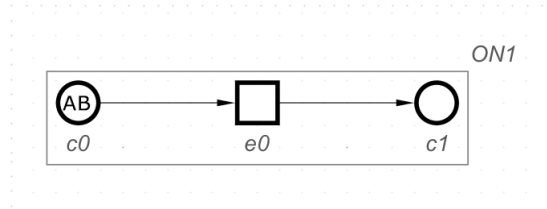


Figure 5.3: Coloured tokens in simple example (Step 1).

In this example, event $e0$ is ready to fire as it is activated both by A and B tokens present inside condition $c0$. As a result of executing coloured event $(e0, A)$, token A moves to condition $c1$. After that, coloured event $(e0, B)$ is still enabled and can fire using token B , as shown in Figure 5.4.

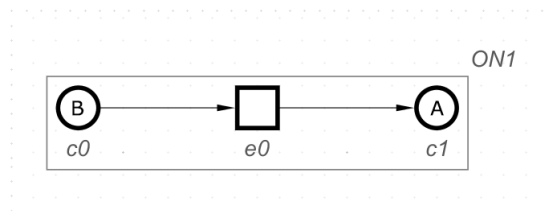


Figure 5.4: Coloured tokens in simple example (Step 2).

Then, coloured event $(e0, B)$ can occur causing token B to move to condition $c1$, as illustrated in Figure 5.5.

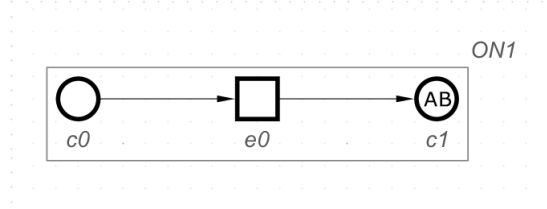


Figure 5.5: Coloured tokens in simple example (Step 3).

Another example is shown in Figure 5.6, where condition $c0$ has two coloured tokens (A and B), both enabling event $e0$.

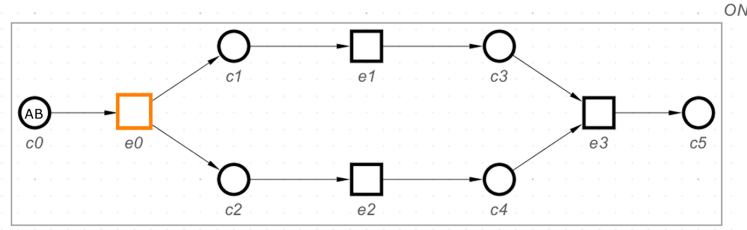


Figure 5.6: Coloured token in ON (Step 1).

As a result of coloured event $(e0, A)$ and $(e0, B)$ occurs, tokens A and B are removed from condition $c0$, and tokens A and B are placed in conditions $c1$ and $c2$. Subsequently, coloured events $(e1, A)$, $(e1, B)$, $(e2, A)$ and $(e2, B)$ are enabled, as shown in Figure 5.7.

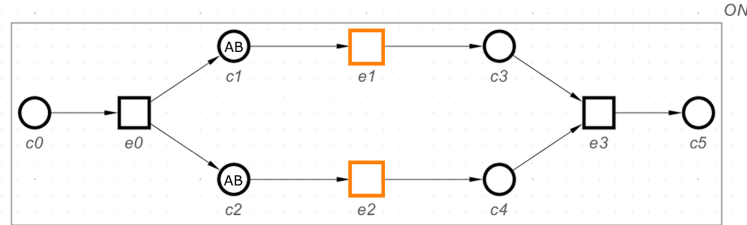


Figure 5.7: Coloured token in one ON (Step2).

As a result of event $(e1, A)$ occurs, token A first moves to condition $c3$, as shown in Figure 5.8.

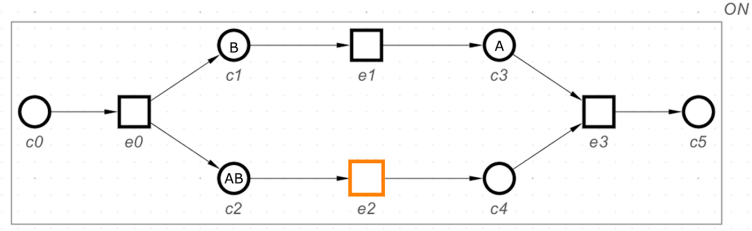


Figure 5.8: Coloured token in one ON (Step 3).

Then, event (e_2, A) occurs, and as a result, token A moves to condition c_4 , as shown in Figure 5.9. Now, event (e_3, A) is ready to be fired. Once this event occurs, token A moves to condition c_5 , as shown in Figure 5.10.

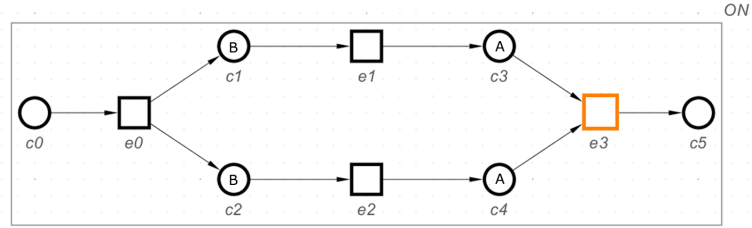


Figure 5.9: Coloured token in one ON (Step 4).

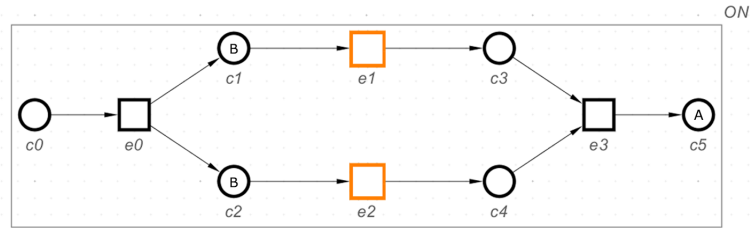


Figure 5.10: Coloured token in one ON (Step 5).

After that, event (e_1, B) occurs, as a result token B moves to condition c_3 , as shown in next figure 5.11.

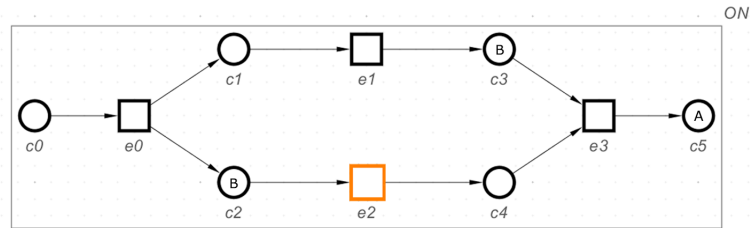


Figure 5.11: Coloured token in one ON (Step 6).

Then, event $(e2, B)$ occurs, and as a result token B moves to condition $c4$, as shown in figure 5.12.

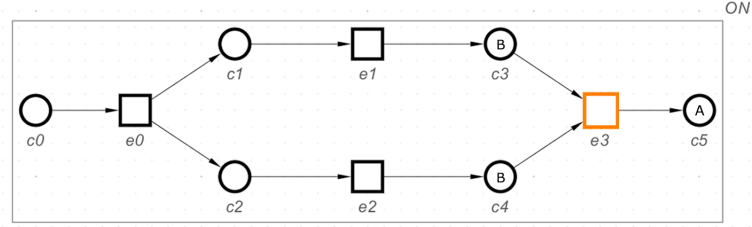


Figure 5.12: Coloured token in one ON (Step 7).

Now event $(e3, B)$ is ready to be fired. Once this event $e3$ occurs, then token B moves to condition $c5$, as shown in figure 5.13.

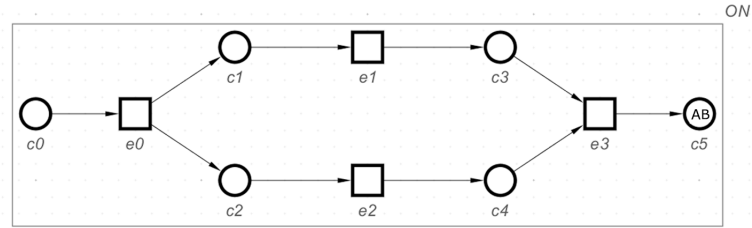


Figure 5.13: Coloured token in one ON (Step 8).

The next example illustrates how two ONs communicate with each other. Initially, events $e0$ in occurrence net $ON1$ and event $e1$ in occurrence net $ON2$ are ready to be fired as coloured events $(e0, A)$ and $(e0, B)$, as shown in Figure 5.14.

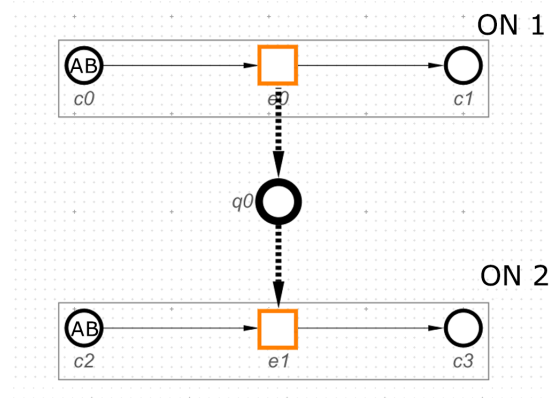


Figure 5.14: Coloured tokens in CCSN (Step 1).

First, coloured events $(e0, B)$ and $(e1, B)$ occur simultaneously. As a result, tokens B in both conditions $c0$ and $c2$ move to conditions $c1$ and $c3$, as shown in Figure 5.15.

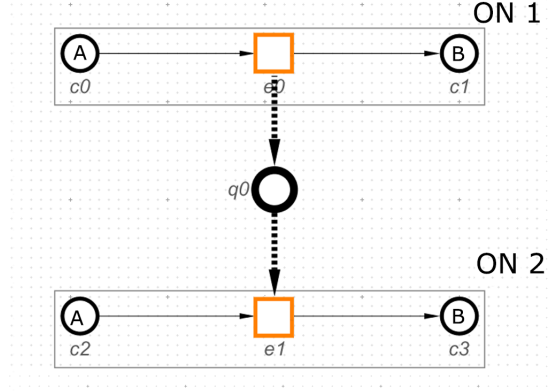


Figure 5.15: Coloured tokens in CCSN (Step 2).

Then, coloured event $(e0, A)$ is ready to be fired. After its execution, token A in condition $c0$ moves to condition $c1$, and token A placed in $q0$ as shown in Figure 5.16.

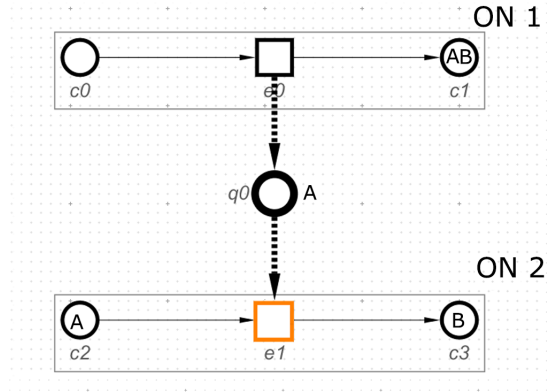


Figure 5.16: Coloured tokens in CCSN (Step 3).

Finally, only event $(e1, A)$ is enabled. After its firing, we obtain marking, as shown in Figure 5.17.

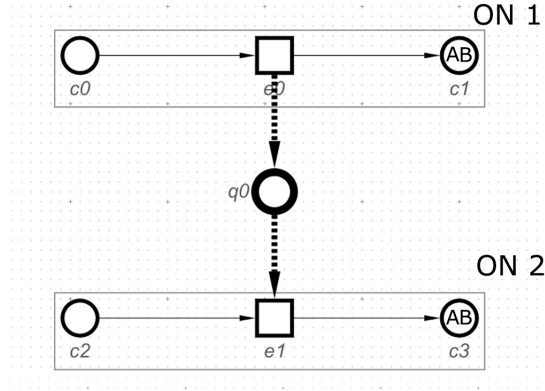


Figure 5.17: Coloured tokens in CCSN (Step 4).

The final example in this chapter demonstrates how the coloured tokens can be used for abstracting large models. As shown in Figure 5.18, four ONs — $ON1$, $ON2$, $ON3$, $ON4$ — represent normal DNS packets.

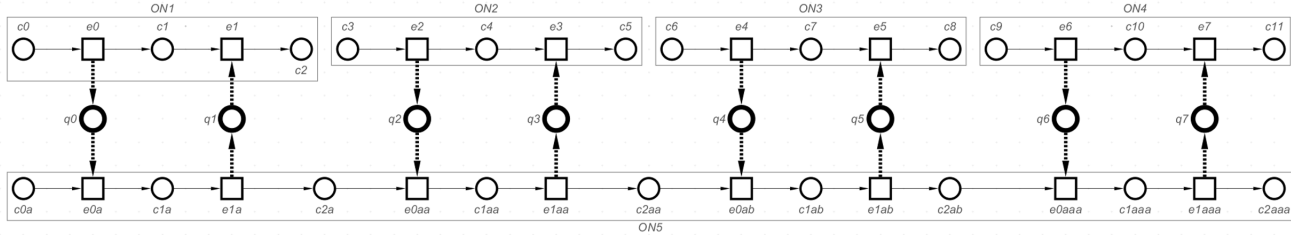


Figure 5.18: Four ONs represent normal packets of DNS and one ON represents Local server.

In $ON1$, which represents one normal packet in DNS, event $e0$ sends a query to event $e0a$, and then the local server $ON5$ responds via event $e1a$. This example assumes there are only four normal packets of DNS, and even then presenting them in one screen is not easy. However, after the coloured extension is applied, the model of Figure 5.18 can be abstracted as one ON ($ON1$) for the four packets of DNS and one ON ($ON2$) for the local DNS server, shown in Figure 5.19.

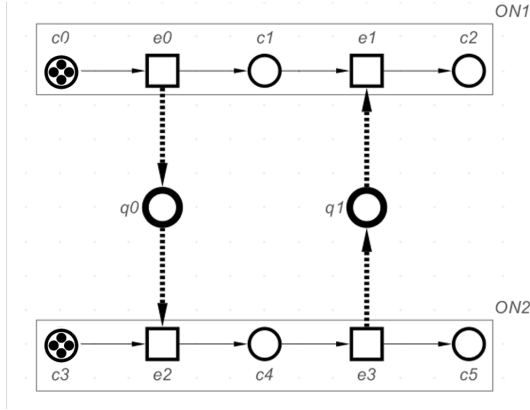


Figure 5.19: Abstracting large model to small one.

Because all four DNS packets are identical and behave in the same way, the ONs are replaced by tokens, and only one ON is obtained for the four ONs, with four tokens for conditions $c0$ and $c1$. Once a new normal packet that behaves in the same way as previous packets is obtained, then the number of tokens in $c0$ and $c3$ is increased by adding more tokens.

The abstraction presented in Figure 5.19 may not be fully satisfactory in all cases as it overapproximates the behaviours of the original CSON. The reason is that in the considered scenario there are four instances of packet behaviour and only one instance of server behaviour. To cope with this situation, we now outline a generalisation of the abstraction described earlier.

We assume that $\text{CSON} = (ON_1, \dots, ON_k, ON, Q, W)$ is such that the following hold (below $m \geq 1$):

- For each ON_i there are distinct buffer places q_1^i, \dots, q_m^i for communicating with ON . Moreover, p_1^i, \dots, p_m^i are distinct events in ON_i , and r_1^i, \dots, r_m^i are distinct events in ON connected to these buffer places in such a way that for $l = 1, \dots, m$, we either have $(p_l^i, q_l^i), (q_l^i, r_l^i) \in W$ (for every $1 \leq i \leq k$), or we have $(r_l^i, q_l^i), (q_l^i, p_l^i) \in W$ (for every $1 \leq i \leq k$).
- The ON_i 's are isomorphic and not communicating with each other. Moreover, p_l^1, \dots, p_l^k are 'isomorphic' events (for every $1 \leq l \leq m$).

- For $i = 1, \dots, k$, the events p_1^i, \dots, p_m^i are causally ordered (in this way) in ON_i .
- The events $r_1^1, \dots, r_m^1, \dots, r_1^k, \dots, r_m^k$ are causally ordered (in this way) in ON .

Then we can model CSON by deleting ON_2, \dots, ON_k together with q_1^i, \dots, q_m^i (for $2 \leq i \leq k$) and attached arcs. Following this we then add (q_l^1, r_l^i) for each deleted $(q_l^i, r_l^i) \in W$, and (r_l^i, q_l^1) for each deleted $(r_l^i, q_l^i) \in W$. Finally, the initial marking of ON_1 is set to have k tokens in each initial condition. Figure 5.20 illustrates the construction for the example in Figure 5.18.

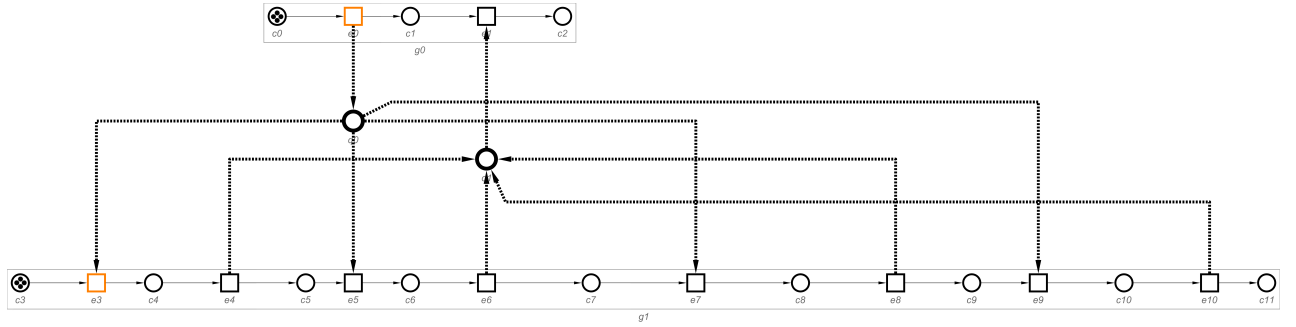


Figure 5.20: Abstracting large model to small one.

5.8 Concluding remarks

This chapter proposed an extension of the notion of CSON to cover situations when more than one concurrent process is executed using the static net structure of CSON. We have proposed this in two different ways, and have emphasised the motivation for this extension. The concept of big data has been presented and discussed. PNs and CPNs and their features also have been covered in this chapter. Examples of the proposed extension have also been provided. Future work will focus on the use of the extension in abstracted versions of SONs, as well as on the application of this extension to real scenarios of crime or cybercrime cases.

Chapter 6

SONCraft: A Plug-in for Loading Large Amount of Data

6.1 Summary

In this chapter, we present the open-source SONCraft plug-in, which is used help users in automatically load large amounts of data. Both the concept and tools are explained Finally, we evaluate the SONCraft plug-in using the standard principles of user interface interactions.

6.2 Introduction

SONCraft provides several features allowing the construction of SON-based models and the validation of such models. Several implemented algorithms allow for a friendly user interface, that support effective usage. Also, using such tools help users construct SON models for different scenarios. SON formalism can be used to make it easy to portray and analyse complicated behaviours, such as failure behaviours of complex evolving systems. Such systems are usually large with their subsystems concurrently communicating with each other. As such, their behaviours can be modified

by other systems. For example, distributed systems, such as software, are regularly updated; such updates can add new features or solve existing issues. Although there is a massive volume of data available on the internet, related to different domains and there are several tools and analysis techniques that deal with such data automatically, SONCraft lacks features that deal with this type of data. To examine how SONCraft can deal with large data sets, we have built an automatic SON model for this particular task. We designed and implemented a plugin that can help users in automatically load large amounts of data. This chapter is organised as follows: Section 7.2 presents SONCraft; Section 7.3 introduces the Loader algorithm; Section 7.4 describes the plugin's implementation and design; Section 7.5 focuses on how the plug-in works; finally, Section 7.6 provides several concluding remarks.

6.3 Loader algorithm

The first step of discovering SONCraft's ability in terms of automatically loading/retrieving data from a particular source such as CSV files was to test how SONCraft deals with this type of data. This plug-in was developed to retrieve data from CSV files and convert all data in that file into usable data for SON models. We created a CSV file with random data. We assumed that each row in the CSV file represents one ON. For instance, as shown in the next diagram, we assumed that $ON1 = (c0, e0, c1)$ was one ON. Moreover, $(c0)$ was the first initial place, $(e0)$ was the first event, and $(c1)$ was another place. We did the same for the following ONs, namely ON2, ON3, and ON4. The labels in the document help illustrate each particular ON, in other words, they are not shown in the CSV file.

So, the loader must consider each row as an ON and model it in SONCraft. Figure 6.1 shows how the result is illustrated.

c0	e0	c1
c2	e1	c3
c4	e2	c5
c6	e3	c7

Table 6.1: CSV file data

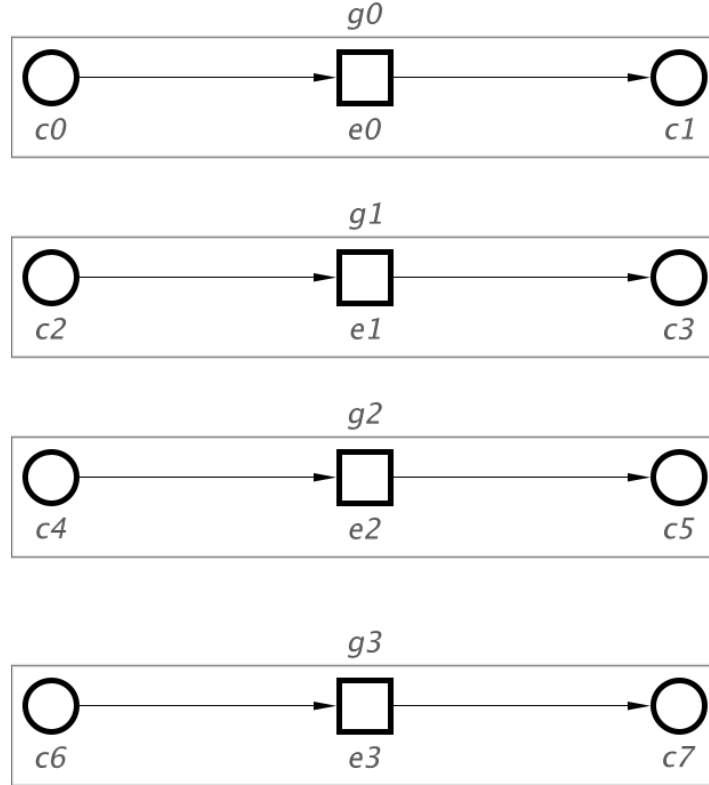


Figure 6.1: Conversion of CSV file data into ONs.

6.3.1 The algorithm

The main idea of the loader algorithm is that the input consists of a CSV file's full path. The algorithm's first step is to generate the initial array which will store sets of ONs. Then, to perform the first iteration over the rows of the Excel sheet. We assume each cell that loaded from the CSV file to be a particular node. Each time, meaning with each iteration, we check whether or not the node has previous nodes, or, if this is not the case, it is identified as the end of the row. Every time, we identify

ALGORITHM 3: Loader Algorithm

Input: file_path : Full path to CSV file

Output: SON = Set Ons, generated from the CSV file.

```
1 Start ;
2 array set_of_ons = [] ;
3 SON final_SON ;
4 for String current_line in file_lines do
5   String [] = current_line_tokenized;
6   Integer index = 0;
7   boolean add_to_existing_on = false;
8   ONElement en_element ;
9   ON current_ON;
10  for (String string_element in current_line_tokenized) do
11    if (index mod 2 == 0) then
12      | en_element = Condition;
13    else
14      | en_element = Event;
15    end
16    if (en_element == Condition && on_element not in
        set_of_ons) then
17      | current_ON;
18      | new_ON ← attach(condition, en_element);
19    else
20      | current_ON ← find(on_element, set_of_ons);
21      | current_ON ← attach(condition, en_element );
22      | add_to_existing_on ← =true;
23    end
24  end
25  if (add_to_existing_on = true) then
26    | set_of_ons ← add(current_ON, set_of_ons);
27  else
28    |
29  end
30  final_SON ← build_from_ONS(set_of_ons);
31 end
```

the node we check if it is a place (odd) or an event (even). Finally, once the entire row is verified, it is built as an ON. To achieve this, we must use an import method that SONS accept, like `InputStream`. The return is a SON model whos main utility is to transform an `InputStream` from a chosen file to a SON model that can be later drawn in the SON Workcraft plugin.

6.4 Plugin design and implementation Design

6.4.1 Graphical User Interface design

This section discusses the plugin design and implementation. First, the GUI (Graphical User Interface): the SONCraft data tool GUI menu classes have two custom classes. The first class is “SONDataTool”, and it’s used for the importing of SON data (“Import SON Data”), as illustrated in Figure 6.3 below.

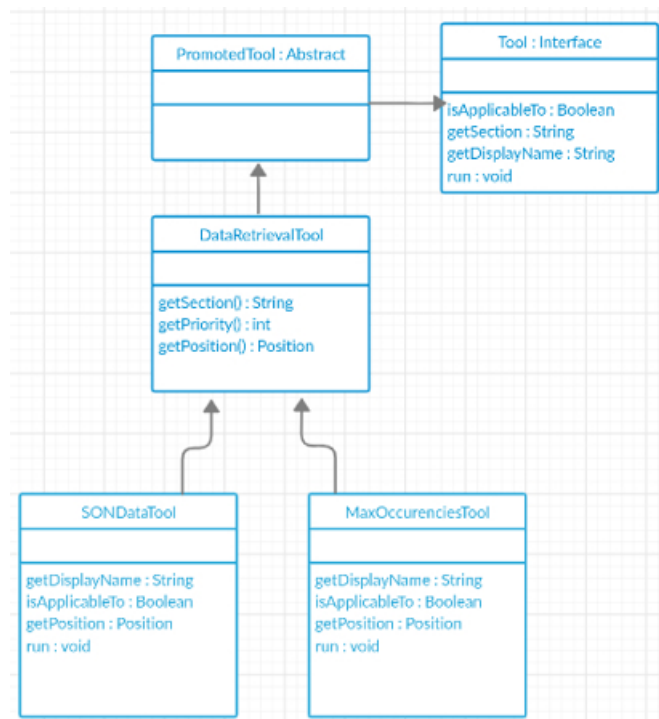


Figure 6.2: Graphical user interface classes.

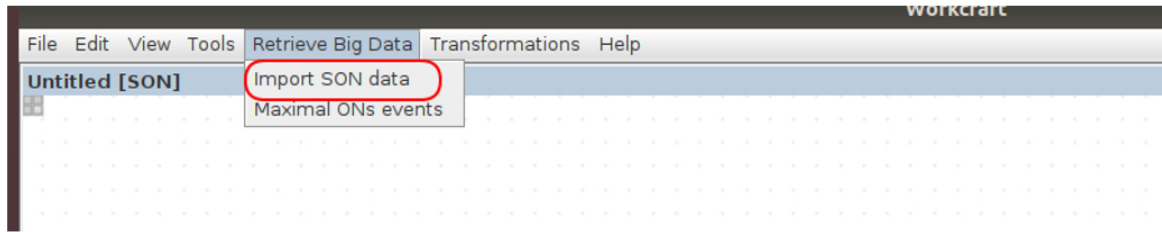


Figure 6.3: Import SON Data tool.

The second class is “MaxOccurenciesTool”, it is used for “Maximal ONs events”, as illustrated in Figure 6.4

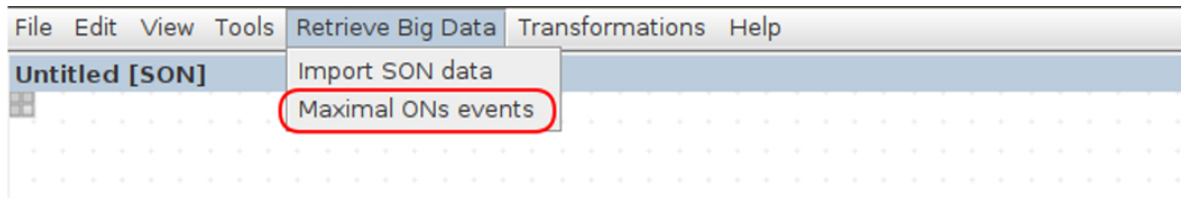


Figure 6.4: Maximal ONs events tool.

Both of these classes extend WorkCraft’s “DataRetrievalTool” class, which is responsible for retrieving data from files. The “DataRetrievalTool” class extends the abstract, “PromotedTool” class, which in turn implements the Interface tool; all these tools are present in the “org.workcraft” package, as part of the “Workcraft-Core” project. The “DataRetrievalTool” and “SonDataTool” classes are part of the “org.workcraft.plugins.transform” package.

6.4.2 Data retrieval Back-end design

The “MaxOccurenciesTransformer” class is present in the “org.workcraft.plugins.son.interop” package and implements the “Transformer” interface which is found in the “org.workcraft.interop” package that also contains the algorithm described in the previous section. This algorithm is also responsible for transforming data into a SON graph. The “Model

transform(Model model)” method is the main method of the “MaxOccurencies” algorithm. It is called from the “org.workcraft.plugins.transform.MaxOccurenciesTool” class, by using the “void run(WorkspaceEntry we)” method. It accepts a SON Model and calculates the “Max Occurencies” output. The “SON importSON(InputStream inputStream)” method is responsible for transforming data into a SON model.

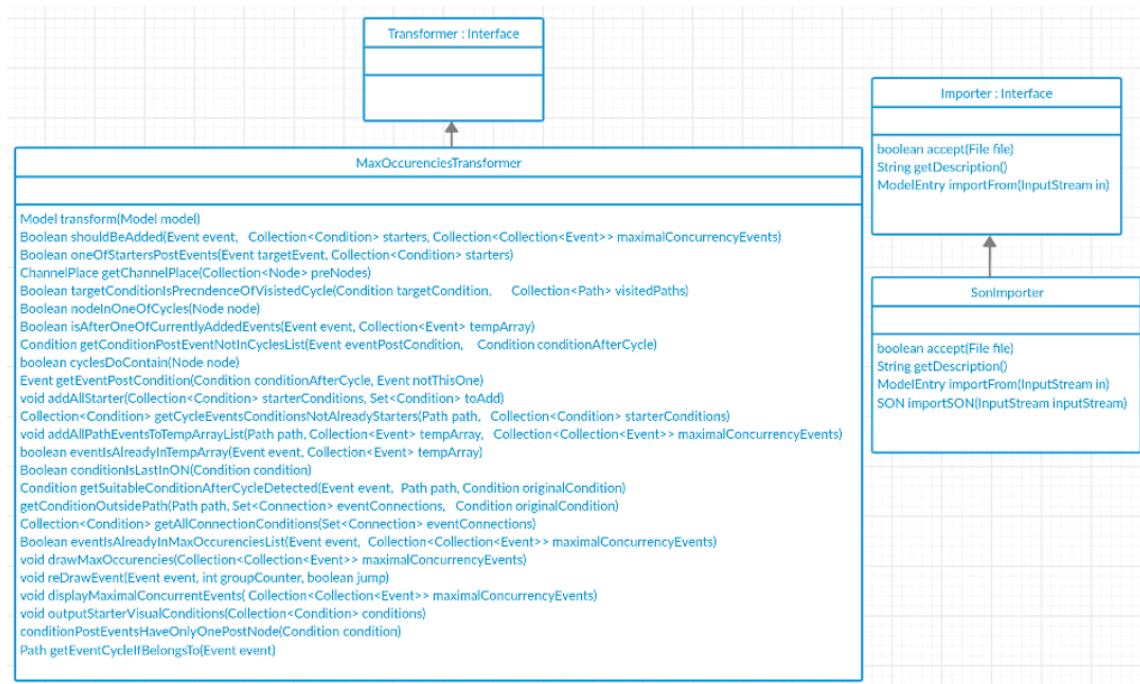


Figure 6.5: Data retrieval feature class diagram.

In figure 6.6 I have display the SON import code and I have added the code’s explanation in green, as Java comments. The import SON method accepts an Input-Stream and returns a SON model, its main utility is to transform an InputStream got from the chosen file to a SON model to be later drawn in the SON Workcraft plugin.


```

1  /* The import SON method accepts an InputStream and returns a SON model, its main
2     utility is to transform an InputStream got
3     from the chosen file to a SON model to be later drawn in the SON Workcraft plugin */
4  public SON importSON(InputStream inputStream) throws DeserialisationException {
5      SON son = null; //Initialisation of the SON model
6      if (inputStream != null) { //Whenever the input stream is not null, do
7          try {
8              //Initialise an XSSWorkbook object from the input stream. The
9              XSSWorkbook object is an object present in the Apache POI library. The
10             apache POI library has been added as an external library to the plugin.
11             XSSWorkbook workbook = new XSSWorkbook(inputStream);
12             //Get the first sheet of the Excel file.
13             XSSFSheet defaultSheet = workbook.getSheetAt(0);
14             //Get a row iterator from the sheet
15             Iterator<Row> rowIterator = defaultSheet.iterator();
16             if (rowIterator != null) {
17                 //Instantiate the son object
18                 son = new SON();
19                 //Iterate over the excel sheet rows
20                 while (rowIterator.hasNext()) {
21                     //Instantiate a POI row each time there is an Excel row.
22                     Row row = rowIterator.next();
23                     // We're going to iterate over the cells now
24                     Iterator<Cell> cellIterator = row.cellIterator();
25                     boolean conditionConstruction = true; //Always begin
26                     with condition
27                     // Each cell is a node, instantiate the Node object.
28                     Node node = null;
29                     // This node has a previous node?
30                     Node previousNode = null;
31                     // Iterate over the row's cell.
32                     while (cellIterator.hasNext()) {
33                         // Get the cell
34                         Cell cell = cellIterator.next();
35                         // Get the cell value
36                         String cellValue = cell.getStringCellValue();
37                         // Instantiate a node from the node text value
38                         node = son.getNodeByReference(cellValue);
39                         if(conditionConstruction){ //
40                             If conditionConstruction is
41                             true and node is null → it is a condition
42                             if(node == null){
43                                 // Instantiate the condition from the cell value
44                                 Condition condition = (Condition)son.createCondition(
45                                 cellValue, null);
46                                 // Set the label of the condition
47                                 condition.setLabel(cellValue);
48                                 node = condition;
49                             }
50                         }else{ // If
51                             conditionConstruction is false and node is null → it is
52                             an
53                             event
54                             if(node == null){
55                                 // Instantiate the event
56                                 Event event = (Event)son.createNode(cellValue, null,
57                                 Event.class);
58                                 // set the event label from cell value
59                                 event.setLabel(cellValue);
60                                 node = event;
61                             }
62                         }
63                     }

```

Figure 6.6: The code.

```

1         if(previousNode != null){ //If it hasn't a previous node,
2             connects previous node to current node
3             son.connect(previousNode, node, Semantics.PNLINE);
4         }
5         if(conditionConstruction){ // Switch back the
6             conditionConsutruction (true → false or false → true)
7             conditionConstruction = false;
8         }else{
9             conditionConstruction = true;
10        }
11        previousNode = node;
12    }
13    }
14    }
15    // Finally close the POI workbook before returning the SON model
16    workbook.close();
17    } catch (Exception exception) {
18        throw new DeserialisationException(exception);
19    }
20    }
21    // Return the SON model
22    return son;
23 }

```

Figure 6.7: Part two of the code

6.5 Evaluating the SONCraft Plug-in

This section evaluates the SONCraft plug-in using principles of user interface interactions. The evaluation will be based on the user interface and will use a well-known concept called Usability Heuristics for User Interface Design by Jakob Nielsen [52].

The first heuristic is the visibility of system status. The SONCraft clearly implements this feature, which tells the user what is occurring while the platform is in use. For instance, the users will always be informed of any tools the SONCraft picked up. Additionally, if some non-logical steps are taken during the creation of the SON model (e.g., if the user tries to link two events directly or connect two places), the plug-in will warn the users. Ideally, the user would be provided with the complete steps taken after he/she finishes building a model. This feature would allow users to trace the steps while constructing the models.

The second heuristic is the match between the system and the real world, and the SONCraft plug-in meets this criterion. For instance, the menu and panels are the same as other similar tools used today. However, some tools do not meet this criterion. For example, the zooming tool in the SONCraft model shows a lack of accuracy, meaning that the user may find it difficult to zoom in on a particular part of the model. In other words, when the user tries to zoom in on part of the model, the zooming tool will move to the far left side instead of focusing on the desired part. Additionally, the shortcuts of some tools do not match the general expectations. For instance, in standard software, the user utilises (Ctrl+ '+') to zoom in and (Ctrl+ '-') to zoom out; however, (Ctrl+ '=') is used to zoom in when using the SONCraft model.

The third heuristic is user control and freedom. This means the user can undo his/her actions if a mistake is made. This feature exists in the SONCraft model; the user can easily undo and redo actions.

Another heuristic of usability is consistency and standards [52], which means

the user does not need to inquire about words, situations or actions while using particular tools. Although the SONCraft plug-in uses different terminologies for parts of the tool, these designations are due the terminologies used for the model (SONs), which include places, events and channel places. However, the main menu meets this heuristic of usability, including common terms such as “file”, “edit”, “view” and “tools”.

The fifth heuristic of usability is error prevention. The system not only provides a warning message to the user but will also first try to prevent the error. SONCraft has this feature; for instance, the user cannot link two events or places with each other.

6.6 Conclusion and Future Work

In this chapter, we presented SONCraft, an open-source platform used to model, analyse, and visualise SONs. SONCraft provides an easy-to-use user interface that helps users in modelling and analysing particular cases, as well as in model validation. SONs and their features within SONCraft are now being extended to new applications.

Also, different tools included in SONCraft were presented, such as editing, structural analysis, simulation, and scenario generation tools, as well as Time SON tools. We have also provided details of the plug-in used to load big data, as well as our solution to the overlapping issues with this plug-in. We have integrated these tools with SONCraft to detect cybercrime cases with a large, complex evolving model. In the future, we plan to allow SONCraft to integrate with large-scale visualisation tools such as Hadoop. The current SONCraft plug-in already appropriately and effectively supports visualisation. However, analysing big data requires some abilities that exist in Hadoop. This kind of integration will allow the platform to be used in the industry, but it can also provide more ability and flexibility when visualising big data in SONCraft.

Chapter 7

Conclusion and Future work

7.1 Conclusion

This thesis presents the extension of the SON framework. For example, this framework improves both handling and visualising data sets involved in cybercrime investigations.

In Chapter 2, the background of SONs and their features were presented. Moreover, the visualisation background, the visualisation stages and visualisation techniques were presented, and the background of the domain name system was discussed.

In Chapter 3, the limitations of visualisation in SONs were described in terms of visualising large data sets. Moreover, the challenges resulting from the overlapping and out-of-order placements of ONs were addressed. Also, a novel solution was proposed to deal with this issue based on a maximal step execution policy. Furthermore, additional algorithms were provided to address such challenges. Finally, a plug-in that deals with this problem was implemented, and results obtained were discussed.

In Chapter 4, a DNS tunnelling attack was investigated. Moreover, the detection of that threat and how it occurred was provided. The SON model and its features for detecting DNS tunnelling in the event of an actual attack were provided, and a

plug-in was implemented to detect this kind of attack.

Chapter 5 reported the motivation of producing new definition in SON, called visualisation and abstract conditions in SON (coloured tokens). The formal definition of this extension was provided and approved in this section. The background of basic Petri nets, coloured Petri nets and big data was discussed in this chapter as well. Finally, we discuss different scenarios and examples of how to apply the new extensions to different examples of SONs.

Chapter 6 presented the SONCraft plug-in which is used help users in automatically load large amounts of data. Finally, we evaluate the SONCraft plug-in using principles of user interface interactions.

7.2 Future work

With regard to future work identified at the end of each chapter, various opportunities exist and some concerns remain to be addressed in the area of SONs. In this section, we enumerate and discuss these possibilities in more detail.

7.2.1 Visualisation of behavioural abstraction through maximal concurrency policy

One of the directions of the future work involves implementing visualisation based on local maximal concurrency [41]. This is a step execution policy that introduces an extension of the basic model of Petri nets by adding the notion of located transitions and locally maximally concurrent executions of co-located transitions to represent the compartmentalisation of membrane systems [41]. The main idea relates to specifying the locality for event, with each event belonging to a fixed unique locality. The precise mechanism to achieve this goal involves introducing a partition of the set of all transitions using locality mapping [9].

7.2.2 Detection of multi-attacks and use of different DNS tunnelling tools

Although we obtained promising results with regard to detection of DNS attacks using SON, the time limitation prevented us from considering several multi-attacks simultaneously. In other words, if more than one attack occurs in very complex scenarios, the problem becomes more interesting. However, the current SONs still need improvement in terms of visualisation. The analysis of more than one attack is necessary to evaluate the ability of SON from two main perspectives. The first concerns multiple levels of SONs. For instance, as seen in Figure 7.1, we focus on two levels of DNS protocol, namely the client side and the local server DNS.

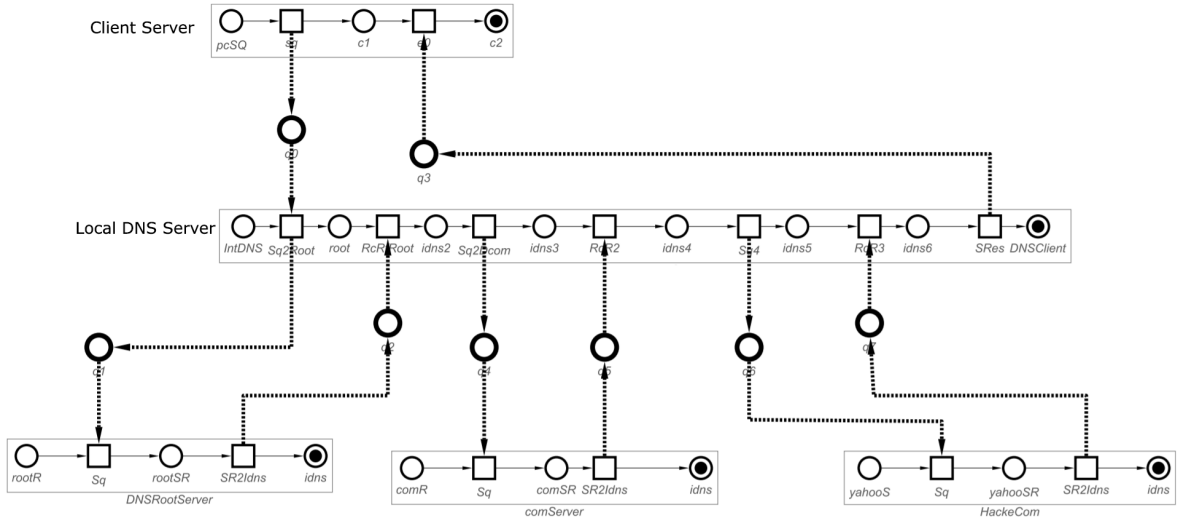


Figure 7.1: SON Model for Normal DNS protocol .

However, as can be seen in Figure 7.1, the DNS protocol has more than one level, such as the DNS root server and the LTD server (e.g., '.com and .net'). Investigations into more than one level of DNS protocols will be very complex and challenging as well. In addition, it is interesting to use DNS tunnelling tools other than 'iodine' to compare various attack behaviours and assess how the algorithm deals with these types of attacks.

7.2.3 Investigation of large data sets using coloured tokens of SONS

Using coloured tokens of SONS will allow modelling and analysis of large, sophisticated data sets — for instance, Twitter, which is used extensively in today’s world. The crime has increased in Twitter and other social media. The size of the data set poses challenges in analysing such platforms. However, we believe that after improving SONS to deal with large amounts of data, one can expect a suitable result with regard to crime detection. Thus, the identification and detection of crime in Twitter and social media continue to pose challenges to investigators. Notably, SONS have provided good results in the cybercrime detection field in chapter 4. The idea is to implement a tool that read the public data on Twitter automatically and convert it into SON models. After that, provide algorithms that can identify any particular crime patterns.

7.2.4 Data mining and SONS

Both data mining and process mining provide exemplary techniques and tools for visualisation. The combined implementation of these technologies and SONS could provide novel analysis or visualisation results. Visualisation and analysis models automated with the features of SONS (which can be achieved via data mining) could outperform manual capture of behaviours and the analysis that follows. Extraction of patterns and knowledge using both data mining and SONS could also prove to be a promising avenue of research.

Bibliography

- [1] Talal Alharbi. Visualising data sets in structured occurrence nets. <https://github.com/talalsm/visualizemaximalevents>, May 2018.
- [2] Talal Alharbi. mix packets script. <https://github.com/talalsm/mix/>, May 2019.
- [3] Talal Alharbi. split packets script. <https://github.com/talalsm/mix/>, May 2019.
- [4] Talal Alharbi. Tunnelling detection using structured occurrence nets (sons). <https://github.com/talalsm/detecttunellingson>, May 2019.
- [5] Eike Best and Brian Randell. A formal model of atomicity in asynchronous systems. *Acta informatica*, 16(1):93–124, 1981.
- [6] A Chris Bogen and David A Dampier. Preparing for large-scale investigations with case domain modeling. In *DFRWS*, 2005.
- [7] Kenton Born and David Gustafson. Detecting dns tunnels using character frequency analysis. *arXiv preprint arXiv:1004.4358*, 2010.
- [8] Andy Cockburn and Bruce McKenzie. An evaluation of cone trees. In *People and Computers XIV—Usability or Else!*, pages 425–436. Springer, 2000.

- [9] Philippe Darondeau, Maciej Koutny, Marta Pietkiewicz-Koutny, and Alex Yakovlev. Synthesis of nets with step firing policies. In *International Conference on Applications and Theory of Petri Nets*, pages 112–131. Springer, 2008.
- [10] Kord Davis. *Ethics of Big Data: Balancing risk and innovation.* ” O’Reilly Media, Inc.”, 2012.
- [11] DeNiSe. Denise is a proof of concept for tunneling tcp over dns in python. <https://github.com/mdornseif/DeNiSe/>, May 2005.
- [12] Alan Dix. *Human-computer interaction.* Springer, 2009.
- [13] DNScapy. Pierre bienaimé. <https://github.com/FedericoCeratto/dnscapy/>, May 2010.
- [14] DNScapy. dnscat2. <https://github.com/iagox86/dnscat2/>, May 2014.
- [15] Ciro Donalek, S George Djorgovski, Alex Cioc, Anwell Wang, Jerry Zhang, Elizabeth Lawler, Stacy Yeh, Ashish Mahabal, Matthew Graham, Andrew Drake, et al. Immersive and collaborative data visualization using virtual reality platforms. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 609–614. IEEE, 2014.
- [16] Erik Ekman. Heyoka. <https://github.com/yarrick/iodine/>, 2014.
- [17] Cheikh Kacfeh Emani, Nadine Cullot, and Christophe Nicolle. Understandable big data: a survey. *Computer science review*, 17:70–81, 2015.
- [18] Greg Farnham and A Atlasis. Detecting dns tunneling. *SANS Institute InfoSec Reading Room*, 9:1–32, 2013.
- [19] J-D Fekete and Catherine Plaisant. Interactive information visualization of a million items. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 117–124. IEEE, 2002.

- [20] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, 2005.
- [21] Heyoka. Heyoka. <http://heyoka.sourceforge.net/>.
- [22] W Hibbard, Haim Levkowitz, Janet Haswell, Penny Rheingans, and Florian Schroeder. Interaction in perceptually-based visualization. In *Perceptual issues in visualization*, pages 23–32. Springer, 1995.
- [23] Harry Hochheiser and Ben Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.
- [24] Derek T Hughes. Smartfiles–ict innovation in complex criminal investigations. *International Journal of Innovative Computing*, 5(1), 2015.
- [25] Martin Husák, Jana Komárková, Elias Bou-Harb, and Pavel Čeleda. Survey of attack projection, prediction, and forecasting in cyber security. *IEEE Communications Surveys & Tutorials*, 21(1):640–660, 2018.
- [26] Alfred Inselberg and Bernard Dimsdale. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In *Proceedings of the 1st conference on Visualization’90*, pages 361–378. IEEE Computer Society Press, 1990.
- [27] Ryszard Janicki and Maciej Koutny. Invariants and paradigms of concurrency theory. In *Parle’91 Parallel Architectures and Languages Europe*, pages 481–496. Springer, 1991.

- [28] Kurt Jensen. A brief introduction to coloured petri nets. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 203–208. Springer, 1997.
- [29] Brian Johnson and Ben Shneiderman. *Tree-maps: A space-filling approach to the visualization of hierarchical information structures*. IEEE, 1991.
- [30] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. Dns performance and the effectiveness of caching. *IEEE/ACM Transactions on networking*, 10(5):589–603, 2002.
- [31] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. Dns performance and the effectiveness of caching. *IEEE/ACM Transactions on networking*, 10(5):589–603, 2002.
- [32] Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.
- [33] Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.
- [34] Eugene A Kapp, Frédéric Schütz, Lisa M Connolly, John A Chakel, Jose E Meza, Christine A Miller, David Fenyo, Jimmy K Eng, Joshua N Adkins, Gilbert S Omenn, et al. An evaluation, comparison, and accurate benchmarking of several publicly available ms/ms search algorithms: sensitivity and specificity analysis. *Proteomics*, 5(13):3475–3490, 2005.
- [35] Eugene A Kapp, Frédéric Schütz, Lisa M Connolly, John A Chakel, Jose E Meza, Christine A Miller, David Fenyo, Jimmy K Eng, Joshua N Adkins, Gilbert S Omenn, et al. An evaluation, comparison, and accurate benchmarking of several publicly available ms/ms search algorithms: sensitivity and specificity analysis. *Proteomics*, 5(13):3475–3490, 2005.

- [36] Daniel A Keim. Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.
- [37] Daniel A Keim. Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.
- [38] Muzammil Khan and Sarwar Shah Khan. Data and information visualization methods, and interactive mechanisms: A survey. *International Journal of Computer Applications*, 34(1):1–14, 2011.
- [39] Seokyeon Kim, Seongmin Jeong, Sung Uk An, Jae Seok Yoo, Sang Min Han, Hanbyul Yeon, Sangbong Yoo, and Yun Jang. Big data visual analytics system for disease pattern analysis. In *Proceedings of the 2015 International Conference on Big Data Applications and Services*, pages 175–179. ACM, 2015.
- [40] HCM Kleijn and Maciej Koutny. Infinite process semantics of inhibitor nets. In *International Conference on Application and Theory of Petri Nets*, pages 282–301. Springer, 2006.
- [41] Jetty HCM Kleijn, Maciej Koutny, and Grzegorz Rozenberg. Towards a petri net semantics for membrane systems. In *International Workshop on Membrane Computing*, pages 292–309. Springer, 2005.
- [42] Maciej Koutny and Brian Randell. Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. *Fundamenta Informaticae*, 97(1-2):41–91, 2009.
- [43] Jensen Kurt. Coloured petri nets: Basic concepts, analysis methods and practical use. *EATCS Monographs on Theoretical Computer Science. 2nd edition, Berlin: Springer-Verlag*, 1997.

- [44] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review*, 35:100219, 2020.
- [45] Bowen Li, Maciej Koutny, and Brian Randell. Soncraft: A tool for construction, simulation and verification of structured occurrence nets. *School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1493*, 2016.
- [46] Bowen Li, Brian Randell, Anirban Bhattacharyya, Talal Alharbi, and Maciej Koutny. Soncraft: A tool for construction, simulation, and analysis of structured occurrence nets. In *2018 18th International Conference on Application of Concurrency to System Design (ACSD)*, pages 70–74. IEEE, 2018.
- [47] Hela Ltifi, Mounir Ben Ayed, Adel M Alimi, and Sophie Lepreux. Survey of information visualization techniques for exploitation in kdd. In *2009 IEEE/ACS International Conference on Computer Systems and Applications*, pages 218–225. IEEE, 2009.
- [48] James Manyika. Big data: The next frontier for innovation, competition, and productivity. http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation, 2011.
- [49] Alessio Merlo, Gianluca Papaleo, Stefano Veneziano, and Maurizio Aiello. A comparative performance evaluation of dns tunneling tools. In *Computational Intelligence in Security for Information Systems*, pages 84–91. Springer, 2011.
- [50] M Michalos, P Tselenti, and SL Nalmpantis. Visualization techniques for large datasets. *Journal of Engineering Science & Technology Review*, 5(1), 2012.
- [51] Paul Mockapetris and Kevin J Dunlap. *Development of the domain name system*, volume 18. ACM, 1988.

- [52] Jakob Nielsen. 10 usability heuristics for user interface design. *Nielsen Norman Group*, 1(1), 1995.
- [53] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information systems*, 24(1):25–46, 1999.
- [54] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79, 1998.
- [55] Peter Pirolli and Ramana Rao. Table lens as a tool for making sense of data. In *Proceedings of the workshop on Advanced visual interfaces*, pages 67–80. ACM, 1996.
- [56] Catherine Plaisant, Brett Milash, Anne Rose, Seth Widoff, and Ben Shneiderman. Lifelines: visualizing personal histories in: *Proceedings of the chi’96 conference on human factors in computing systems*, 1996.
- [57] Brian Randell. Occurrence nets then and now: the path to structured occurrence nets. In *International Conference on Application and Theory of Petri Nets and Concurrency*, pages 1–16. Springer, 2011.
- [58] Brian Randell. Occurrence nets then and now: the path to structured occurrence nets. In *International Conference on Application and Theory of Petri Nets and Concurrency*, pages 1–16. Springer, 2011.
- [59] Brian Randell and Maciej Koutny. Failures: their definition, modelling and analysis. In *International Colloquium on Theoretical Aspects of Computing*, pages 260–274. Springer, 2007.

- [60] Brian Randell and Maciej Koutny. Structured occurrence nets: Incomplete, contradictory and uncertain failure evidence. *School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1170*, 2009.
- [61] George G Robertson, Jock D Mackinlay, and Stuart K Card. Cone trees: Animated 3d visualizations of hierarchical information. In *CHI*, volume 91, pages 189–194, 1991.
- [62] A. Romanovsky. A study of atomic action schemes intended for standard Ada. *Journal of Systems and Software*, pages 29–44, 1998.
- [63] Ben Shneiderman. Inventing discovery tools: combining information visualization with data mining. *Information visualization*, 1(1):5–12, 2002.
- [64] Robert Spence. *Information visualization*, volume 1. Springer, 2001.
- [65] Shan Suthaharan. Big data classification: Problems and challenges in network intrusion prediction with machine learning. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):70–73, 2014.
- [66] Kees Van Hee, Natalia Sidorova, and Marc Voorhoeve. Soundness and separability of workflow nets in the stepwise refinement approach. In *International Conference on Application and Theory of Petri Nets*, pages 337–356. Springer, 2003.
- [67] Tom van Leijenhorst, Kwan-Wu Chin, and Darryn Lowe. On the viability and performance of dns tunneling. *International Conference on Information Technology and Applications*, 2008.
- [68] Lidong Wang, Guanghui Wang, and Cheryl Ann Alexander. Big data and visualization: methods, challenges and technology progress. *Digital Technologies*, 1(1):33–38, 2015.

- [69] Colin Ware. *Information visualization: perception for design*. Elsevier, 2012.
- [70] Hsu-Chun Yen. Introduction to petri net theory. *Recent Advances in Formal Languages and Applications*, 25(343-373):70, 2006.
- [71] Ji Soo Yi, Youn ah Kang, and John Stasko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE transactions on visualization and computer graphics*, 13(6):1224–1231, 2007.
- [72] Ji Soo Yi, Youn-ah Kang, John T Stasko, and Julie A Jacko. Understanding and characterizing insights: how do people gain insights using information visualization? In *Proceedings of the 2008 Workshop on BEyond time and errors: novel evaluation methods for Information Visualization*, page 4. ACM, 2008.